(4)

AR-005-710

MRL-GD-0023

AD-A218 430

DTIC FILE COPY

DTIC
ELECTE
FEB 26 1990
S D
Co B

FOR PUBLIC RELEASE

90 02 23 124

THE PROCEDURES FOR EASY RUNNING OF THE HULL COMPUTER CODE UNDER VMS

Ross J. Kummer

MRL General Document
MRL-GD-0023

ABSTRACT

Version 121 of the HULL computer code has been installed on a VAX 8700 at MRL. This report explains the complex command procedures that were written to provide an easy interface for running HULL on the VAX system. Full instruction guides are included, with installation details to provide easy implementation of the procedures at other VAX sites.

Published by DSTO Materials Research Laboratory
Cordite Avenue, Maribyrnong, Victoria 3032, Australia
Telephone: (03) 319 3887
Fax: (03) 318 4536

AR No. 005-710

Contents

THE PROCEDURES FOR EASY RUNNING OF THE HULL COMPUTER CODE UNDER VMS

1. INTRODUCTION

The HULL code [1] is a group of public domain computer programs to solve hydrodynamic flow problems. The Hull system comprises a library of code from which Fortran programs are generated, a program called SAIL [2] that is used to maintain this library, and a program called PLANK that is a preprocesser for HULL. The HULL system is capable of running on a variety of computers and is currently installed at a number of sites throughout the world.

The HULL system produces code for hydrodynamic problems. Version 121 can solve two or three dimensional problems in either Eulerian mode, Lagrangian mode, or a combination of both Eulerian and Lagrangian modes. HULL also has the capability to produce a variety of plots of the data produced by the calculations.

The HULL code is written in standard FORTRAN but must be altered to run on different computers due to such things as varying bit sizes of variables and the different file opening statements for different machines. This is achieved by setting the variables that SAIL accepts (either in the input file to SAIL or installed in the HULL library) so that SAIL extracts the code from the HULL library specific to the machine being used. The Materials Research Laboratory (MRL) has been running HULL on a CYBER 205 supercomputer [3], but version 121 of HULL has also been installed on a VAX 8700 computer at MRL.

This report presents Digital command language (DCL) procedures which have been written with the aim of managing the great number of files produced by the HULL system when running problems, and to control the sequence of programs that are run under the HULL system. The DCL procedures have been written with portability in mind so that they could be used at any VAX site running VMS 4.7 or later, with minimal alteration. This report describes the logic and use of these procedures.

2. THE HULL SYSTEM

2.1 HULL components

The HULL system comprises:

- A library of subroutines and sequences of code from which programs are generated. The library holds all the code necessary to generate the Fortran code to solve calculations, generate grids and plot results.
- The SAIL program, which is used to extract code from the library.
- The PLANK program, which generates the input to the SAIL program.

Solving any problem with the HULL system requires that KEEL be run to generate the problem grid and parameters . Then the program HULL must be run to cycle through time to solve the problem generated by KEEL. Plots can be produced from KEEL and HULL runs in a variety of forms, such as contour and histogram plots . The PULL program generates plots from the information for elements or cells, and STATION generates plots from data collected at designated points fixed either in space or relative to material flow (stations). Figure 1 represents the organisation of the HULL system, and Table 1 presents the terms used in the HULL system.

Table 1 Glossary of terms used in describing HULL 121 and its installation

HULL	Program to solve the difference equations and time iterations (cycler). Also, loosely, the HULL 121 system.
HULL-JOB	File containing DCL commands.
HULL LIBRARY	File containing entire HULL 121 system in packed form.
HULL 121	Version #121 of the entire HULL system.
KEEL	Program to define and fill the computational grid (pre-processor).
MATLIB	File containing material property constants.
PLANK	Program to expand user input and generate secondary input for SAIL.
PRIMARY INPUT	Input for KEEL, HULL or PULL generated by the user.
STATION	Program to generate graphical output for grid particle and station information.
PULL	Program to process HULL results and generate graphical output (post-processor).
SAIL	Utility to manage HULL 121, operating on coded internal identifiers and directives.
SECONDARY INPUT	Input for SAIL to generate appropriate version of KEEL, HULL or PULL from HULL LIBRARY.
USER INPUT	See PRIMARY INPUT.

2.2 Host requirements

The HULL system is written such that it can be installed on a variety of machines. This is achieved by setting flags on input to SAIL, which instruct SAIL to include code specific to each machine from the HULL library. For convenience, a new library may be generated with these flags included so that any further SAIL run will generate code according to the pre-set flags. The installation of HULL on the VAX 8700 involved generating a new HULL library with a SAIL run, with the approriate SAIL input definitions for a VAX 8700 machine. From the new library a new PLANK program was generated, as well as a materials library. Once the new HULL system library was generated, all code produced was automatically VAX 8700 machine compatible.

3. DESIGN OF HULL COMMAND FILE

3.1 The command file

The command file (HULL_JOB) is written in DCL, and performs the following tasks:

- Assigns the files to be used for a calculation to the names that HULL recognises so that files generated have meaningful names.
- Performs the run phase according to the requirement of the user.
- Maintains VAX sub-directories so that several calculations can co-exist on the system without confusion.
- Allows for multi-user access to commonly used files, whilst allowing individual users to set their own parameters.

Figure 2 is a functional description of the command file.

3.2 File handling

HULL produces default outputs, such as SAIL.DAT and OUTPUT.DAT from a SAIL run. The command file renames or defines the default outputs to give them meaningful names from generic roots. For example, the problem identifier is used so that the source listing for KEEL produced by SAIL, called SAIL.DAT, is renamed as KEEL999.FOR (assuming a problem identifier of 999). Some typical files produced are:

DAYFILE.999	-	file from command file to show sequence of events
DUMPFILES.DAT	-	holds extensions of dump files to be plotted. If /REPEAT is used, it is re-read to obtain extensions
HULL999.DUMP4	-	FOR004.DAT file is defined to this file from HULL
HULL999.DUMP9	-	FOR009.DAT file is defined to this file from HULL. HULL999.DUMP9 holds the station information from a run
KEEL999.DUMP4	-	FOR004.DAT is defined to this file from KEEL
KEEL999.DUMP9	-	FOR009.DAT is defined to this file from KEEL. KEEL999.DUMP holds the station definitions
*prog*999.EXE	-	the executable file for the main run phase
*prog*999.FOR	-	the source file for the main run phase
*prog*999.OUT	-	the normal printed output from a run
INPUT.999	-	the program input is copied to this file every run
INPUT2.DAT	-	this file is generated by PLANK as input for SAIL
PULLHULL.PLOT	-	plot output for a PULL run of HULL

PULLKEEL.PLOT	-	plot output for a PULL run of KEEL
STATHULL.PLOT	-	plot output for a STATION run of HULL
PLANK*prog*999.DIAG	-	diagnostics from a PLANK run
SAIL*prog*999.DIAG	-	diagnostics from a SAIL run

In the above *"prog"* is either HULL, KEEL, PULLHULL, PULLKEEL, STATHULL, REZONE. The problem identifier used above is 999.

3.3 Parameter setting

The command file is written such that individual users set their own parameters to indicate such things as where files produced by a run will reside, or whether they desire screen-based monitoring. These parameters are assigned in an independent command file called HULL.COM.

3.4 Logic checking

There are several levels of checking in the HULL_JOB command file. They range from confirming the existence of necessary files, to checking that user input to the command file is a valid option.

3.5 Code Structure and Features

The HULL_JOB command file was written in DCL, using subroutines to handle the various functions involved in running the program. Appendix A is a full listing of the command file. Full use is made of VAX operating system lexical functions (e.g. F$ELEMENT(...), F$SEARCH(..) etc.) along with utility type commands as in the case of the SORT command. The program is fully documented internally to assist modification of the DCL.

3.5.1 HULL run phases

Each run of HULL is in three main phases:-

1. Running PLANK to process HULL input and produce expanded secondary input for SAIL.
2. Running SAIL to extract code from the HULL library to produce the final Fortran program.
3. Running the final program; either KEEL, HULL, PULL or STATION.

Each phase coresponds to a subroutine i.e. RUN_PLANK, RUN_SAIL and RUN_PROG respectively. Each phase begins with the definition of the input and output files. The names that are given to output files generally are a result of these definitions. All defined files are deassigned at the end of each phase or if an error occurs. Necessary files for each phase are also accounted for at the start of each phase by a call to subroutine CHECK.

3.5.2 Utility subroutines

Several subroutines are shared between larger subroutines. The values passed to these subroutines are either passed by placing input to them on the same line as the call to the subroutine e.g. CALL CHECK "PLANK_EXE" , or by placing the value passed into a global symbol that is then referenced in the subroutine. A list of these subroutines follows, with a short explanation of each:

ERROR	-	Writes a message to the terminal and dayfile for non-fatal warning errors.
WRITMESS	-	Writes messages to screen and dayfile.
ERROR_COM	-	Called when a fatal error occurs (e.g. on CONTROL_Y). It performs such functions as renaming or closing files.
FINISH	-	On command file completion this subroutine closes some possibly open files and does a general "clean up" before the command file is stopped.
CHECK	-	Checks files for existence. If they are not found, a fatal error has occurred.
GETCPU	-	Recovers resource information for the run.
PUT_LINE	-	Puts line of text on a VT100 compatible terminal screen in inverse video.
FORMAT_SCREEN	-	Clears screen and writes blank status line.
SCROLL_TOP	-	Sets scroll region to top part of screen.
SCROLL_BOTTOM	-	Sets scroll region to bottom part of screen.

3.5.3 User input subroutines

Each possible type of user input to the command procedure has a subroutine to handle the logic of accepting valid user input. A list of these subroutines follows, with a short explanation of each:

SET_BATCH	-	Batch queue parameters are accepted, e.g. the name of the queue and cost code for batch file.
INTER_IDENT	-	An identifier is prompted for, and directory created if necesary. The string is kept and included in file names.
GET_DUMPS	-	The string representing the dump file versions is accepted and processed to create a file holding the extensions for the files.
GET_PROG	-	The string containing the required HULL system phases to be run is accepted, and checked against a list of valid options.
GET_MAIN_INPUT	-	The name of the input file for the HULL run is obtained for, then the file checked for existence.

4. USERS GUIDE

4.1 Installation

There are two main files involved with running the command procedure. The files are HULL_JOB.COM which contains the actual DCL code to run HULL, and HULL.COM which sets up an individual user's definitions. There should be no need to change HULL JOB.COM in everyday circumstances, and this file should be installed in a directory accessible to all users. Each user should install a copy of HULL.COM in their own root

directory (see below) so that individual parameters can be set. An explanation of the two files follows.

4.1.1 HULL_JOB.COM (system level installation)

HULL_JOB handles the actual running of HULL and may reside in any directory, but should ideally reside in the directory of one user (the HULL System Manager). In this case the file can be modified and the modifications are then accessible to all users. The file could simply be executed by typing the command:

"@[directory-name]HULL_JOB ", but could be set to :
"$ RUN_HULL:==@[directory-name]HULL_JOB" in HULL.COM, so the command is "RUN_HULL"

4.1.2 HULL.COM (user definitions)

HULL.COM sets up definitions for individual users, and makes assignments to variables to be later used in the HULL_JOB file. This file is automatically executed at the beginning of each run of the HULL_JOB command file. In addition HULL.COM should be executed at the start of each interactive session, most easily achieved by including an execution statement in the individual users LOGIN.COM. The file must reside in the user defined root directory (see below). A typical HULL.COM listing appears in Appendix B. Some of the assignments required are listed below (lower case names are user dependent):

$ ROOT_DIR:==user.sub-directory

Defines the directory in which HULL.COM must reside for each user.

$ WORK_DISK:==disk-name:

Defines the disk name for the WORK Directory area. If the default disk is to be used then the assignment is WORK_DISK:=="".

$ WORK_DIR:==user.work-directory

This is the name of the directory in which the main input files for the HULL system must reside, and from where sub-directories for each problem will be created.

$ RUN_HULL:==@[directory-name]HULL_JOB

The directory name is the directory in which HULL_JOB.COM resides. This can be another user's area. The command to run the command file is then "$ RUN_HULL".

$ DEFINE HULL_LIB [directory-name]hull-library

The HULL library is referenced by using HULL_LIB without having to refer to directory paths*. The library can reside in any other users area.

$ DEFINE PLANK_EXE [directory-name]PLANK.EXE

PLANK is referenced using PLANK_EXE without having to refer to directory paths*. PLANK.EXE can reside in any other users area.

$ DEFINE SAIL_EXE [directory-name]SAIL.EXE

SAIL is run by simply typing "RUN SAIL_EXE", where SAIL.EXE can reside in any users area*.

$ DEFINE MAT_LIB [directory-name]materials-library

The materials library can be held in a common user accessible area and referenced as MAT_LIB*.

$ DEFINE DAYFILE ['WORK_DIR']DAYFILE.DAT;

The file holding the messages from the command file can be set to exactly the line above. At the completion of HULL_JOB this file is then copied from the Work directory to the subdirectory for the problem being run.

$ PLOT_OBJ:=="[directory-name]plot-package.obj"

PLOT_OBJ is assigned to the full name of the plotting package that is to be linked with a PULL or STATION program. This may be an object file for primitive CALCOMP routines or drivers for particular graphic devices. This assignment may be set to "" if the plotting option used does not require any files to be linked with PULL or STATION.

$ VT100==1

Set VT100=1 for VT100 compatible terminals (e.g. VT100, VT220, VT240 etc) to indicate screen formatting is to be done, VT100=0 for non vt100 terminals, and VT100=3 to run the command procedure without the help displays or screen formatting.

$ COST_LIST:==number1/number2/../numberN

Assign COST_LIST:=="" if a cost code is not required at top of batch files. The first element in the list is later used as the default menu option for the cost number at the top of the batch command file.

$ BATCH_LIST:==name1/name2/../nameN

BATCH_LIST holds list of batch names for the system, which can be set to "" if only one default batch queue is ever to be used. The first element in the list is later used as the default menu option for the batch queue name.

$ LINKHULL :== LINK/link-option1.../link-optionN

This linking option is used to link the object file for KEEL, HULL etc. A typical assignment might be "LINKHULL:== LINK/MAP".

$ COMPILEHULL :== FORTRAN/compile-option1/.../compile-optionN

This compiler option is used to compile the source code for KEEL, HULL etc. A typical assignment might be "COMPILEHULL:== FORTRAN/CROSS/LIS", or simply "COMPILEHULL:== FORTRAN ".

$ MONITOR_RUN:== spawned process before main run

MONITOR_RUN is typically "MONITOR/PROCESS TOPCPU" but can be set to "".

Files that can be shared between several users, namely SAIL, PLANK, the HULL library, the MATLIB, and the HULL_JOB command procedure, could reside in the user directory of a HULL manager, or at the VAX system level and accessed by other users via references in HULL.COM to the respective areas in which they lie. Appendix C includes a typical directory set-up for a HULL system.

4.2 How to run HULL

4.2.1 Screen based organisation and help

There are three main menus in the command procedure which prompt for the necessary data to run the command file. The first menu allows a user to specify the phases that are to be run (see Table 2). The next menu allows the main input file to KEEL, HULL, PULL, or STATION to be specified. Following this is a menu to obtain a unique problem identifier for the run. If either PULL or STATION is to be run, a menu will prompt for the version numbers of the dump files from which to generate plots. If /BATCH is specified in the first directory, a menu will prompt for the parameters required for the batch run.

These menus are represented in Appendix D. The inputs that would be received from the first four menus can be placed on the same line as the call to the command file, in the order of their respective menus, e.g.

$ RUN_HULL PULLKEEL KDAT999.DAT 999 1-3

which translates to

@HULL_JOB generate PULL data file identifier dumpfile version

4.2.2 Interactive use of menus

4.2.2.1 Menu 1 - Options to run several combinations of phases

If KEEL or HULL is to be run then KEEL or HULL is specified as the input. If a PULL or STATION run is required to generate plots from either KEEL or HULL dump files, then either PULLKEEL, PULLHULL or STATHULL is specified. By themselves, these commands will run PLANK then SAIL then the actual KEEL, HULL, PULL or STATION

program generated. With the option /SAIL, the SAIL will be run without first running PLANK. The option /MAIN will begin the sequnce at the stage of compiling the main program to be run. /MAINEXE will assume a .EXE file exists, and start the sequence by running the main .EXE file. The option /NOEXE will stop the sequence when the .FOR file of the main program is generated. Diagramatically, the various sequences are as shown in Table 2.

These options accomodate a variety of needs. For instance, if it is required that the source code for KEEL is to be generated, then edited, then the program actually run, KEEL/NOEXE could be specified, the resulting .FOR file edited, then KEEL/MAIN specified in the menu. If a restart is required for a HULL run, simply specify HULL/MAINEXE, where the program will automatically accept the last dump file written as the cycle at which it is to restart.

Table 2 Menu 1 - Run Phases

INPUT	SEQUENCE OF EVENTS
prog	run PLANK->run SAIL->compile *prog*->link *prog*->run *prog*
prog/PLANK	run PLANK
prog/SAIL	run SAIL->compile *prog*->link *prog*->run *prog*
prog/MAIN	compile *prog*->link *prog*-> run *prog*
prog/MAINEXE	run *prog*
prog/NOEXE	run PLANK->run SAIL
prog/SAIL/NOEXE	run SAIL

"prog" is either KEEL, HULL, REZONE, PULLKEEL, PULLHULL or STATHULL.

A further option is /BATCH, which specifies that after accepting all other options, a process is spawned to run the command file in batch mode. The particular batch queue required is prompted for, along with all other relevant options. A message to the screen will indicate when the batch job is finished. The log file for the job will be held in the subdirectory for the problem run and its contents appended to the dayfile for the problem. If all defaults are to be used for the batch run then /DEFBATCH should be specified, in which case no menu will appear.

4.2.2.2 Menu 2 - Inputting data to be used with the main program being run

Any HULL program to be run requires the normal HULL input file. For use by the command file, this input file must reside in the Work directory (i.e. the area defined as WORK_DIR in HULL.COM). If the file name does not exist, the correct file name will be re-prompted for. Any changes to the original file required for further runs must be made

in the Work directory. A copy of the input file will be made to the sub-directory for each particular run.

4.2.2.3 Menu 3 - The problem indentifier

An identifier is required to provide a unique label for a problem. As a result, a sub-directory of the Work directory will be set up to hold all files generated from a problem, and any file generated will have the problem identifier as part of its name. The sub-directory created for a problem will be called RUN_"ident".DIR i.e. for problem 999, RUN999.DIR is created. If an identifier is given for which a sub-directory already exists, all files in the directory are assumed to belong to this problem.

4.2.2.4 Menu 4 - Dump file specification

If PULL or STATION is to be run, then this menu will accept the version numbers of either the KEEL, or HULL dump files, where these dump files have extensions .DUMP4 for PULL runs, and .DUMP9 for STATION runs. The version numbers are expressed in terms of ranges, i.e. 4-END or 3-6 etc. , where several ranges can be specified by separating each with commas. This is illustrated with the following group of files.

```
HULL999.DUMP;1
HULL999.DUMP;2
HULL999.DUMP;3
HULL999.DUMP;4
HULL999.DUMP;5
HULL999.DUMP;6
HULL999.DUMP;7
HULL999.DUMP;8
```

To specify that the first three dumps are to be plotted plus every second dump, the input line could be "1-3,4,6,8". Also, the last "n" plots can be requested by typing "LASTn", and every n^{th} dump file can be plotted by stating "SKIPn".

4.2.3 Running without menus for experienced users

If the command file help menus are not required, the variable VT00 in HULL.COM can assigned as "$ VT100 == 3" (see section 4.1.2). This will produce the prompts for each option if required, and any error messages, but will not display the help menus or provide any screen formatting. The inputs that would be received from the menus can be placed on the same line as the call to the command file, in the order of their respective menus, e.g.

```
$ RUN_HULL      PULLKEEL    KDAT999.DAT     999             1-3
```

which specifies:

```
@HULL_JOB   generate PULL    data file     identifier   dumpfile version(s)
```

If there is an error in any of these specifications, an error message will be displayed, and a prompt will be made for a new value.

4.2.4 Batch run options

If /BATCH is used as a qualifier to the specification of the run phase (either in the first menu or as part of the command line) then a prompt will be made for the parameters to run the program in batch mode. Valid options are displayed as a list previously assigned in HULL.COM (see section 4.1.2) . The values prompted for are the name of the system queue that the job will be run on, and the string that is to appear at the top of the command file that will be used to run the job. Default options will be displayed, and are accepted by pressing return at the relevant prompt(s). If it is known that the default values are to be used, the qualifier /DEFBATCH can replace /BATCH, in which case no prompts for values are made. A typical command line might be:

```
$ RUN_HULL    HULL/MAINEXE/DEFBATCH
 (@HULL_JOB) (run executable HULL in batch mode with default options)
```

4.3 How to make changes to the command file

Typical changes that individual installations may want to make to the command procedure are as follows.

4.3.1 The processing of plots

When PULL or STATION has created plot files called either PULLKEEL.PLOT, PULLHULL.PLOT or STATHULL.PLOT, a subroutine call to PROCESS_PLOTS is made. This subroutine currently copies all files with "PLOT" extensions created since start of execution of the command file, to a file called FOR016.DAT. If a different operation is to be performed to the plot files, for example a command to print the file to a plotting device, this could best be performed in this subroutine.

4.3.2 "Clean-Up" operations

Near the end of the command procedure, a subroutine called FINISH is called. If particular operations are desired to be performed at the completion of the command file, they could be placed in this subroutine. Typical operations to be performed are the purging or deletion of files, or the printing of the HULL run diagnostics. The subroutine has a no-abort command at the start ("SET NOON") which is switched off at the end ("SET ON"). This permits processing to continue even if commands which would normally cause the command procedure to exit through fatal errors are encountered, e.g. attempting to delete non-existent files.

5. CONCLUSIONS/SUMMARY

The HULL code is a group of programs to solve Hydrodynamic problems in Eularian and Lagrangion modes.

Running HULL on a VAX system involves a complex sequence of events, comprising the running of many individual programs. Considerable expertise is required to successfully execute the correct sequence of programs and manage the multitude of

input and output files required for each program. The command procedure described in this report reduces what is a complex and difficult task down to a straightforward operation. This procedure provides a "user-interface" for running HULL, which caters for the less experienced users of the HULL system and/or the VAX operating system, but also allows for concise commands by the experienced user. All file/house-keeping and similar functions are performed transparently to the user and the execution of complex sequence of programs is reduced to simple meaningful commands. The procedures are easily portable to other sites running VMS, and allow for a variety of VAX site configurations, including clustered systems.

6. ACKNOWLEDGEMENTS

The author would like to thank David Smith of MRL for liberal use of his time and resources in explaining the operation of the HULL system, and assistance in writing this paper.

7. AVAILABILITY OF THE COMMAND PROCEDURE

An up to date version of these procedures may be obtained with this report for a nominal fee.

8. REFERENCES

1. Matuska, D.A. and Osborn, J.J. "Hull Documentation", Orlando Technology Inc., Florida, 1985.

2. "Sail User's Guide", Shalimar Research and Technology Inc., Florida, 1983.

3. Smith, David L. "JCL procedures to run the HULL Code on the CYBER 205 Computer Installed on CSIRONET", MRL-TN-508, Nov 1986.

Appendix A: Listing of the Command Procedure

```
$! The command file sets the default dir to "work_dir" where
$! this has been defined in the hull.com file . The initial
$! data file to run HULL must be present in this directory.
$! Once the prob. identifier has been accepted a new sub-directory
$! will be created named run + identifier .
$! REQUIRED PARAMETERS :
$! PROG_TO_RUN  or P1 - holds the program to be run i.e. HULL , KEEL etc
$! MAIN_INPUT   or P2 - holds the name of the main input file
$! IDENT        or P3 - holds the problem identifier
$! DUMPS        or P4 - holds the dump file option for plotting
$! Hence if the program was to be run in batch mode , or run using
$! one input line  e.g. @main_input  KEEL KDAT70.DAT 70 , this would
$! create subdirectory .RUN70 , then generate and run KEEL.
$!
$! The settings that must be made in HULL.COM are
$! ROOT_DIR:==="directory in which HULL.COM will reside"
$! WORK_DISK:==="disk in which the work directory resides"
$! WORK_DIR:==="directory in which data files for HULL input reside"
$! DEFINE/NOLOG HULL_LIB        "full name of HULL library with directory path"
$! DEFINE/NOLOG PLANK_EXE       " full name of executable PLANK program"
$! DEFINE/NOLOG SAIL_EXE        " name of SAIL program"
$! DEFINE/NOLOG MAT_LIB         " name of MATERIALS LIBRARY file"
$! PLOT_OBJ:==  "name of plotting object file to be linked with PULL or STATION"
$! set VT100=1 for VT100 compatible terminals or VT100=0 if not
$! VT100==0
$! procname:==  " arbitrary name ,to be used as a process name"
$! linkhull :==  "linking option for HULL .e.g. LINK/NOMAP"
$! compilehull:==  "compilation option for HULL , e.g. FOR/NOLIS"
$! monitor_run:==  "what process is spawned for run , e.g. MONITOR PROC/TOPC"
$! run_hull :==  " @name of this file with directory path"
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ DEFINE/NOLOG  SYS$ERROR     ERRORS.DAT
$ DEFINE/NOLOG  DAYFILE       'WORK_DISK'['WORK_DIR']dayfile.dat;
$ CALL ASSIGN_VALS
$ TYPE SYS$INPUT

                EXECUTING HULL.COM
-------------------------------------------------------------------
$!==========================================================
$ SET DEFAULT 'ROOT_DISK'['ROOT_DIR']
$ @HULL                         ! go to directory holding HULL.COM then execute
$ CALL ASSIGN_VALS   ! it to make sure defaults values are set.
$ CALL PAGE                     ! clear the screen
$!==========================================================
$!========================== open dayfile ==================
$ DAYFILE:==DAYFILE.DAT                         !real name of DAYFILE for run
$ OPEN/WRITE/ERR=ERROR_F MESSF DAYFILE          !open dayfile
$ GOTO CONT0                                    ! continue if open successfull
$ ERROR_F:                                      ! else
$ WRITE SYS$OUTPUT " ERROR  OPENING DAYFILE.    ! give error message
$ CONT0:                                        !continue
$!---------------------------------------------
$ CALL SCROLL_BOTTOM
$ SET DEFAULT 'WORK_DISK'['WORK_DIR']   ! go to work dir , (to get input file)
$ PROG_TO_RUN:=='P1'                            ! get possible input from command lin
```

```
$ MAIN_INPUT:==`P2`
$ IDENT:==`P3`
$!==================== check existence of necessary files ======
$                                       ! make sure defined files are
$ CALL CHECK "PLANK_EXE"        ! in existence
$                                       ! necessary files are
$ CALL CHECK "MAT_LIB"                  ! materials file
$                                       ! hull library
$ CALL CHECK "HULL_LIB"                 ! plank program
$!========================= ask for name of program to run ======
$ CALL GET_PROG                         ! get name of prog
$                                       ! i.e. HULL, PULL, KEEL/MAINEXE etc
$ IF .NOT.DEBUG THEN GOTO NOBUG
$ MONITOR_RUN:==""
$ COMPILEHULL:==""`COMPILEHULL`/NOOP/DEBUG"
$ LINKHULL:==""`LINKHULL`/DEBUG"
$NOBUG:
$!========================  get main input file ============
$ CALL GET_MAIN_INPUT                                   ! get user input for hull
$ MOD:==""`F$EXTRACT(4,10,PROG)`"       ! mod may be HULL or KEEL
$ PART1:==""`F$EXTRACT(0,4,PROG)`"      ! part1 may be HULL,KEEL,PULL,STAT
$ IF PROG.EQS."REZONE" THEN PART1:=="HULL"
$!========================= get prob. ident. ================
$!  call relevant subroutine for interactive or batch mode
$!  to receive ident. If in batch mode and error occurs the run is
$!  aborted .
$!
$ IF (INTERACTIVE) THEN CALL INTER_IDENT
$ IF (.NOT.INTERACTIVE) THEN CALL BATCH_IDENT
$!=================================================================
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ SET DEFAULT [.RUN`IDENT`]             ! set default to new sub-dir
$ COPY [-]`MAIN_INPUT` INPUT.`IDENT`;   ! copy main input file to subdirectory
$ COPY [-]`MAIN_INPUT` *.*;             !, to file of same name and to a file
$!-------------------------------------------------------------
$ IF PART1.NES."PULL".AND.PART1.NES."STAT" THEN GOTO NOPULL
$ DUMPS:=""`P4`"        ! if pull or station then possible 4th
$ BQ:=""`P5`"                   ! 5th and 6th inputs are dumps
$ BC:==""`P6`"                  ! extensions,batch que and batch cost
$
$ CALL GET_DUMPS ""`DUMPS`"     ! store extensions of dump files
$ GOTO CONTP                    !to dumpfiles.dat then read back
$NOPULL:
$ BQ:=""`P4`"                   ! if no pull then possible 4th and 5th
$ BC:==""`P5`"                  ! inputs are batch queue and cost
$CONTP:
$!-------------------------------------------------------------
$ IF ((RUNBATCH.EQS."").OR.(.NOT.INTERACTIVE)) THEN GOTO NOBATCH
$ CALL SET_BATCH ""`BQ`" ""`BC`"        ! pass queue and cost to subroutine
$ IF (INTERACTIVE) THEN CALL RECORD
$ GOTO END_RUN
$ NOBATCH:
$!========================= now run problem ================
$!
$ IF (INTERACTIVE) THEN CALL RECORD
$ IF (EXT.EQS."SAIL") THEN GOTO START_SAIL  ! if starting at sail or the
```

```
$ IF (EXT.EQS."MAIN".OR.EXT.EQS."MAINEXE") THEN GOTO START_PROG
$                                                       ! main run , jump to the
$ START_PLANK:                                          ! relevant call to subroutines.
$               CALL RUN_PLANK                          ! run PLANK
$ IF (EXT.EQS."PLANK") THEN GOTO END_RUN!end session if not running prog
$ START_SAIL:
$               CALL RUN_SAIL                           ! run SAIL
$ IF (NOEXE.EQS."NOEXE") THEN GOTO END_RUN !end session if not running prog
$ START_PROG:
$               CALL RUN_PROG                           ! run program
$!==========================================================
$ END_RUN:
$ CALL FINISH                   ! ***END OF COMMAND FILE ***
$!
```

```
$!*********************************************************************
$!                                  SUBROUTINE ASSIGN_VALS :
$! Global values are assigned
$!*********************************************************************
$ ASSIGN_VALS : SUBROUTINE
$ NODISPLAY==0
$! VT100==0
$ PROCNAME=="HULLRUN"+F$CVTIME("``F$TIME()`"",,"HUNDREDTH")
$ TIME=F$TIME()                                          ! get sytem time
$ BEGIN_TIME:=='F$CVTIME(TIME)`
$ SINCE_TIME:=='F$EXTRACT(12,5,TIME)`    ! shorten this time to just hours+mins
$ INTERACTIVE==0                         ! INTERACTIVE=0 for batch
$ IF (F$MODE().EQS."INTERACTIVE") THEN INTERACTIVE==1
$                              !INTERACTIVE=1 for "INTERACTIVE"
$ IF (.NOT.INTERACTIVE) THEN VT100==3
$! VT100=3 indicates no help menus
$ IF (VT100.EQ.3)   THEN NODISPLAY==1
$!
$ MOD:=="
$ PART1:=="
$ ENDSUBROUTINE
$!
```

```
$!*********************************************************************
$!                                SUBROUTINE RECORD :
$!
$! REQUIRED PARAMETERS :
$!*********************************************************************
$ RECORD: SUBROUTINE
$!= = = = = = = = = = = = = = = = = = = = = = = = = open problem-file = = = = = = = = = = =
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CALL PAGE
$ OPEN/READ INTERM SYS$COMMAND
$ IF F$SEARCH("``WORK_DISK`[``WORK_DIR`]RECORDFILE.``IDENT`").NES."" THEN -
   GOTO CONTFF
$CONFNF:
$ OPEN/WRITE/ERR=ERROR_PF RECF `WORK_DISK`[`WORK_DIR`]RECORDFILE.`IDENT`
$ OUTR:=""
$ OUTR[3,28]:=" PHASE RUN"
$ OUTR[32,50]:=" INPUT FILE"
$ OUTR[51,76]:=" DATE/TIME"
$ WRITE RECF OUTR
$ WRITE RECF -
 "   ------------------------------------------------------------"
$ GOTO CONTPF                                                    ! continue if open succe
$CONTFF:
$ OPEN/ERR=ERROR_PF/APPEND RECF `WORK_DISK`[`WORK_DIR`]RECORDFILE.`IDENT`
$CONTPF:                                                         ! continue
$ OUTR:=""
$ OUTR[3,28]:=`BATCH_STR`
$ OUTR[32,50]:=`MAIN_INPUT`
$ OUTR[51,76]:=`BEGIN_TIME`
$ OUTR2=OUTR
$ WRITE RECF "``F$EXTRACT(0,78,OUTR2)`"
$ IF .NOT.RECORD THEN GOTO FINR
$ IF .NOT.INTERACTIVE THEN GOTO FINR
$ADDR:
$ WRITE SYS$OUTPUT " REMEMBER , Press return, with blank line, to finish "
$ADDR2:
$ READ/ERROR=FINR/PROM="Comment >>" INTERM RLINE
$! INQUIRE/nopunct RLINE "Comment >>"
$ IF RLINE.EQS."" THEN GOTO FINR
$! OUTR:=""
$! OUTR[3,73]:="``RLINE`"
$ WRITE RECF "   ``RLINE`"
$ GOTO ADDR2
$FINR:
$ WRITE RECF -
 "   ------------------------------------------------------------"
$ IF F$GETDVI("RECF","EXISTS") THEN CLOSE RECF
$ IF F$GETDVI("INTERM","EXISTS") THEN CLOSE INTERM
$ GOTO CONTC                                                     ! continue if open succe
$ ERROR_PF:                                                      ! else
$ WRITE SYS$OUTPUT " ERROR  OPENING RECORDFILE"                  ! give error message
$CONTC:
$ENDSUBROUTINE
$!-------------------------  ----------------------
$!
```

```
$!*********************************************************************
$!                                   SUBROUTINE SET_BATCH :
$! This subroutine reads in the problem identifier in non_interactive mode
$! A sub-directory is created for this problem if non exists
$! If an error occurs an error message is written to the dayfile
$! and the command file is exited
$!
$! REQUIRED PARAMETERS : IDENT
$!*********************************************************************
$ SET_BATCH : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CALL PAGE
$ COM1:=`P1`            ! get queue name and cost
$ COM7:=`P2`
$ OPEN/WRITE BFILE BATCHFILE.COM        ! open batchfile
$! OPEN/WRITE SFILE SPAWNFILE.COM
$ SAYB := WRITE BFILE                   ! simplify "write" command
$! SAYS := WRITE SFILE
$ LOGFILE=="``WORK_DISK``F$DIRECTORY()`LOGFILE.``IDENT``"
$ DEFQUEUE=F$ELEMENT(0,"/",BATCH_LIST)          ! default queue is first in list
$ DEFCOST=F$ELEMENT(0,"/",COST_LIST)    ! default cost is first in list
$ IF (RUNBATCH.EQS."DEFBATCH") THEN COM1=DEFQUEUE ! if DEFBATCH secified then
$ IF (RUNBATCH.EQS."DEFBATCH") THEN COM7=DEFCOST  ! use the default values
$ IF (RUNBATCH.EQS."DEFBATCH") THEN GOTO DEFS
C!*****************************************
$ CALL SCROLL_BOTTOM
$ IF (NODISPLAY) THEN GOTO NODISP
$ TYPE SYS$INPUT
                BATCH    QUEUE
               ====================

 Input the specific batch queue you wish to use , or hit
      RETURN for default indicated. The log file for the run
      will be held in the directory for the run.
$ NODISP:
$ CALL SCROLL_TOP
$ TOP1:
$ CALL WRITMESS "Valid list is ``batch_list``" "3"  ! display valid list
$ IF COM1.EQS."" THEN INQUIRE COM1 -
  " WHAT IS THE QUEUE (default is ``DEFQUEUE`) ? "
$ IF COM1.EQS."" THEN COM1=DEFQUEUE   !if return use default
$ ILOOP=-1
$TOPB:
$ ILOOP=ILOOP+1
$ TESTSTR=F$ELEMENT(ILOOP," ",BATCH_LIST)          ! check input against each
$ IF TESTSTR.EQS."/" THEN GOTO ERRB                ! element in list
$ IF COM1.EQS.TESTSTR THEN GOTO CONT1
$ GOTO TOPB
$ERRB:
$ CALL WRITMESS " ERROR Incorrect option ``com1``"
$ COM1:=""
$ GOTO TOP1
$ CONT1:
C!*****************************************
$ IF COST_LIST.EQS."" THEN GOTO CONT7
$ CALL PAGE
```

```
$ CALL SCROLL_TOP
$ TOP7:
$ CALL WRITMESS "Valid list is ``COST_LIST`  " "3"
$ IF COM7.EQS."" THEN INQUIRE COM7 -
 " WHAT IS THE COST CODE ( default is ``DEFCOST`) ? "
$ IF COM7.EQS."" THEN COM7=DEFCOST
$ ILOOP=-1
$TOP7B:
$ ILOOP=ILOOP+1
$ TESTSTR=F$ELEMENT(ILOOP,"/",COST_LIST)
$ IF TESTSTR.EQS."/" THEN GOTO ERR7B
$ IF COM7.EQS.TESTSTR THEN GOTO CONT7A
$ GOTO TOP7B
$ERR7B:
$ CALL WRITMESS " ERROR  invalid code ``com7`"
$ COM7:=""
$ GOTO TOP7
$ CONT7A:
$ DEFS:
$ SAYB "``COM7`"
$ CONT7:
C!*****************************************
$ SAYB "$ SET NOVERIFY"                                  ! write contents to batch file
$ SAYB "$ SET DEF ``ROOT_DISK`[``ROOT_DIR`]"
$ SAYB "$ @HULL"
$ PNAME=PART1+"_"+F$EXTRACT(0,6,IDENT)-"["-"]"
$! PNAME=F$EXTRACT(0,6,"``F$USER()`")+"_"+PART1+"_"+F$EXTRACT(0,4,IDENT)-"["-"]
$ SAYB "$TOP:"
$ SAYB "$ ON ERROR THEN GOTO WAIT01"
$ SAYB "$ SET PROCESS/NAME=``PNAME`"
$ SAYB "$ GOTO CONTN"
$ SAYB "$WAIT01:"
$ SAYB "$ WAIT 00:02"
$ SAYB "$ GOTO TOP"
$ SAYB "$ CONTN:"
$ SAYLINE= "$ RUN HULL ``BATCH_STR` ``NEW_INPUT` ``IDENT`"
$ IF PART1.EQS."PULL".OR.PART1.EQS."STAT" THEN SAYLINE=SAYLINE + " REPEAT"
$ SAYB "``SAYLINE`"
$  IF F$GETDVI("BFILE","EXISTS") THEN CLOSE BFILE
$! SAYS -
$!"$ SUBMIT/QUEUE=``COM1`/NOPRINT/NAME=BATCHRUN/LOG=``LOGFILE`/NOTIF BAT
$! SAYS "$ SYNCH/QUEUE=``COM1` BATCHRUN "
$! SAYS "$ TYPE SYS$INPUT "
$! SAYS ""
$!  IF F$GETDVI("SFILE","EXISTS") THEN CLOSE SFILE
$! SPAWN/NOWAIT @SPAWNFILE
$ COM=" SUBMIT"
$ IF COM1.NES."" THEN COM=COM+"/QUE=``COM1`"
$ COM=COM+"/NOPRIN/LOG=``LOGFILE`/NOTIF BATCHFILE.COM "
$ 'COM'
$ CALL WRITMESS " BATCH JOB NOW SUBMITTED
$ ENDSUBROUTINE
```

```
$!*********************************************************************
$!                                SUBROUTINE BATCH_IDENT :
$! This subroutine reads in the problem identifier in non_interactive mode
$! A sub-directory is created for this problem if non exists
$! If an error occurs an error message is written to the dayfile
$! and the command file is exited
$!
$! REQUIRED PARAMETERS : IDENT
$!*********************************************************************
$ BATCH_IDENT: SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ IF (IDENT.NES."") THEN GOTO CONT1      ! if ident passed to P1
$                                        ! then continue
$ MESS:==" ``PROG`_GEN STOPped because prob. ident . missing" ! else call -
$ CALL  ERROR                                                  ! error routin
$ CONT1:
$ FILE=F$SEARCH("RUN``ident`.DIR")       ! check if sub-directory
$ IF (FILE.EQS."") THEN GOTO CONT2       ! previously exists
$ CALL WRITMESS " WARNINIG - subdirectory RUN``IDENT` already exists"
$!                                       ! if so give -
$                                        ! warning -
$ GOTO CONT3                             ! then continue
$ CONT2:                                 ! else create -
$ CREATE/DIR/VERSION=0 [.RUN`IDENT`]     ! sub-directory
$ FILE=F$SEARCH("RUN``ident`.DIR")       ! then check to
$ IF (FILE.NES."") THEN GOTO CONT3       ! see that the
$                                        ! .dir file exists
$ CALL ERROR " ``PROG`_GEN STOPped because subdirectory couldn't be created"
$ CONT3:
$ RET
$ ENDSUBROUTINE
$!                              end-of-subroutine
$!
```

```
$!*****************************************************************************
$!                              SUBROUTINE INTER_IDENT :
$! This subroutine get problem identifier in interactive mode
$! This subroutine acts in much the smae way as BATCH_IDENT  except the
$! user is re-promted for identifier if an error occurs
$!
$! REQUIRED PARAMETERS : IDENT
$!*****************************************************************************
$!
$ INTER_IDENT: SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CALL PAGE                                          ! clear the screen
$ CALL DRAW_LINES                              ! format the screen
$ CALL SCROLL_BOTTOM
$ IF (NODISPLAY) THEN GOTO NODISP
$ TYPE SYS$INPUT
        = = = = = = = = = = = = = = PROBLEM IDENTIFIER = = = = = = = = = = = = = = =

      The problem identifier can be any combination of digits and
    alphabetic charaters . A sub-directory will be created from the
    WORK directory call RUN"ident" . and files created for this problem
    will have the problem identifier appended to their name

$ NODISP:
$ CALL SCROLL_TOP
$!
$! If identifier was not passed to command file with call then prompt for ident.
$!
$ IF (IDENT.EQS."")THEN INQUIRE ident "What is the problem identification "
$ FILE=F$SEARCH("RUN``ident`.DIR")                   ! check if sub-dir
$ TOP:                                                          ! previously created
$ IF FILE.EQS."" THEN GOTO CONT2                     ! if so ,
$ CALL WRITMESS         " WARNING subdirectory already exists"! give warning
$ GOTO CONT3                                                    ! then exit subroutine
$ CONT2:                                             ! else
$ CREATE/DIR/VER=0 [.RUN`IDENT`]                     ! create sub-dir
$ FILE=F$SEARCH("RUN``ident`.DIR")                   ! check if .dir exists
$ IF (FILE.NES."") THEN GOTO CONT3                   !   if so continue
$ CALL WRITMESS "               SUBDIRECTORY COULDN'T BE CREATED"
$ INQUIRE ident " > >ERROR< < What is the prob. ident. "   ! else re-prompt
$ FILE=F$SEARCH("RUN``ident`.DIR")                   ! then start again.
$ GOTO TOP
$ CONT3:
$ IDENT: = = `IDENT`    ! store ident to ident
$ RET                           ! using the global assignment
$ ENDSUBROUTINE                 ! ":= ="
$!                              end-of-subroutine
$!
```

```
$!*****************************************************************************
$!                                  SUBROUTINE RUN_PLANK :
$!The program plank is run ,where the executable version is defined as PLANK_EXE
$!
$! REQUIRED PARAMETERS : NEW_INPUT,IDENT
$! INPUT FILES : INPUT."ident".FOR060.DAT
$! OUTPUT FILES: OUTPUT.DAT,INPUT2.DAT
$!*****************************************************************************
$ RUN_PLANK : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CALL CHECK "''NEW_INPUT'"                    ! check main input file exists
$!-------------------------------------------------------
$ DEFINE/NOLOG      FOR060                     MAT_LIB
$ DEFINE/NOLOG      FOR005                     "''NEW_INPUT'"
$ DEFINE/NOLOG      FOR007                     "INPUT2.DAT;"
$ DEFINE/NOLOG      FOR006                     "PLANK''PROG''IDENT'.DIAG"
$ IF PART1_EQS."HULL".OR.PART1_EQS."KEEL"  THEN GOTO CONTA
$! if generating HULL plots use HULL dump4 file , similarly for KEEL
$ DEFINE/NOLOG FOR004 'MOD''IDENT'.DUMP4;0
$ GOTO CONTB
$ CONTA:
$! to generate HULL or KEEL always use KEEL dump4 file
$ DEFINE/NOLOG      FOR004                     KEEL'IDENT'.DUMP4
$ CONTB:
$!-------------------------------------------------------
$ ON ERROR THEN GOTO ERROR_PLANK !if error in run then do error commands
$ CALL WRITMESS " RUNNING PLANK FOR PROB. ''PROG''IDENT'"
$ RUN PLANK_EXE                                ! run plank
$ CALL WRITMESS " PLANK RUN FINISHED" ! write message to screen
$ ON ERROR THEN CALL ERROR_COM                 ! set error back to general routine
$ GOTO CONT2                                   !   if successful end subrotine
$ ERROR_PLANK:                                 !   else perform error commands i.e.
$ CALL DEASS_PLANK              !   deassign defined files
$                                              !   create appropriate error message
$ CALL ERROR "ERROR IN RUNNING PLANK"                !   call error routine .
$ CONT2:                        !
$ CALL DEASS_PLANK              ! if successful deassign files
$ RET                                          ! then finish subroutine
$ ENDSUBROUTINE
$!*****************************************************************************
$ DEASS_PLANK : SUBROUTINE
$! deassign files used for PLANK run
$!
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CONTA:
$ DEASSIGN FOR060
$ DEASSIGN FOR005
$ DEASSIGN FOR006
$ DEASSIGN FOR007
$ DEASSIGN FOR004
$ RET
$ENDSUBROUTINE
$!
```

```
$!*****************************************************************************
$!                                SUBROUTINE RUN_SAIL :
$! Run the executable version of SAIL . predefined as SAIL_EXE
$!
$! REQUIRED PARAMETERS :ROOT_DIR
$! INPUT FILES : INPUT2,INPUT.OLD
$! OUTPUT FILES: SAIL.DAT,OUTPUT.DAT
$!*****************************************************************************
$ RUN_SAIL : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ NOINPUT=0
$!***********
$! Fix up input.dat so that if CHAGES is defined (i.e. the change deck)
$! then any input.dat is assured to have the line
$! *READ CHANGES
$ IF F$SEARCH("CHANGES").EQS."" THEN GOTO SKIPIT
$OPEN/WRITE OUT INPUT.TEMP
$ IF F$SEARCH("INPUT.DAT").NES."" THEN GOTO CONTI
$ NOINPUT=1
$ WRITE OUT "SAIL"
$ GOTO ADDLINE
$CONTI:
$OPEN/READ IN INPUT.DAT
$ FOUND=0
$TOP:
$ READ/END=ENDF IN REC1
$ WRITE OUT REC1
$ IF F$LOCATE("*READ",REC1).EQ.F$LENGTH(REC1) THEN GOTO TOP
$ FOUND=1
$ENDF:
$ IF FOUND THEN GOTO FOUNDIT
$ADDLINE:
$ WRITE OUT "*READ CHANGES"
$! IF .NOT.NOINPUT THEN CLOSE IN
$ IF F$GETDVI("IN","EXISTS") THEN CLOSE IN
$ IF F$GETDVI("OUT","EXISTS") THEN CLOSE OUT
$! CLOSE OUT
$ CONVERT INPUT.TEMP INPUT.DAT;
$ DELETE INPUT.TEMP;
$ GOTO SKIPIT
$FOUNDIT:
$ IF F$GETDVI("IN","EXISTS") THEN CLOSE IN
$ IF F$GETDVI("OUT","EXISTS") THEN CLOSE OUT
$! CLOSE IN
$! CLOSE OUT
$ DELETE INPUT.TEMP;
$SKIPIT:
$!***********
$ CALL CHECK "INPUT2.DAT"
$!-------------------------------------------------------------+
$ DEFINE NOLOG  OLD HULL.LIB
$!-------------------------------------------------------------+
$ ON ERROR THEN GOTO ERROR_SAIL          !if error in run then special commands
$ CALL WRITMESS " RUNNING SAIL FOR PROB. ''PROG''IDENT'"
$ RUN  SAIL_EXE                          ! run the  sail program
$ CALL WRITMESS " SAIL RUN FINISHED"
```

```
$!  if successful then rename SAIL.DAT as the new .FOR file
$ IF (F$SEARCH("SAIL.DAT").NES."") THEN RENAME SAIL.DAT `PROG``IDENT`.FOR
$!  if successful then rename OUTPUT.DAT as a file containing diagnostics
$IF(F$SEARCH("OUTPUT.DAT").NES."")THEN  RENAME OUTPUT.DAT SAIL`PROG``IDENT`.
$ ON ERROR THEN CALL ERROR_COM
$ GOTO CONT2
$ ERROR_SAIL:                                   ! if error in run then -
$ CALL DEASS_SAIL                       ! deassign files for run
$ CALL ERROR " ERROR IN RUNNING SAIL"   ! call error routine
$ CONT2:
$ CALL DEASS_SAIL                       ! if successful then deassign
$ RET                                           ! files then return
$ ENDSUBROUTINE
$!*****************************************************************************
$ DEASS_SAIL : SUBROUTINE
$! deassign files for sail run
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ DEASSIGN OLD
$ RET
$ ENDSUBROUTINE
```

```
$!*****************************************************************************
$!                                SUBROUTINE RUN_PROG
$! The main module is run . i.e. either HULL . KEEL or PULL
$!
$! COMPILEHULL & LINKHULL are the pre-assigned compiling and linking options
$! for the program (currently assigned in the HULL.COM file)
$! REQUIRED PARAMETERS: PROG,IDENT,INTERACTIVE,COMPILEHULL,LINKHULL
$!
$!*****************************************************************************
$ RUN_PROG : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CALL PAGE                                     ! clear the screen
$!-----------------------------------------------
$! CALL CHECK "INPUT.DAT"
$! CALL CHECK "INPUT2.DAT"
$! CALL CHECK "PLOT_OBJ"
$!-----------------------------------------------
$ DEFINE/NOLOG      FOR060          MAT_LIB
$ DEFINE/NOLOG      FOR005          INPUT.'IDENT'
$ DEFINE/NOLOG      FOR006          'PROG''IDENT'.OUT
$!
$! if HULL or KEEL OR REZONE run then jump over special code for plotting
$!
$ IF PART1.EQS."HULL".OR.PART1.EQS."KEEL"  THEN GOTO DEFPROG
$ FILE9=F$SEARCH("''MOD''IDENT'.DUMP9;")
$ FILE4=F$SEARCH("''MOD''IDENT'.DUMP4;")
$ IF PART1.EQS."PULL" THEN GOTO CONTP
$!
$! if station run then rename dump9 files as for009.dat files
$!
$ IF FILE9.NES."" THEN  RENAME 'MOD''IDENT'.DUMP9;* FOR004.DAT;*
$ GOTO CONTB
$ CONTP:
$!
$! if pull run then rename dump4 files as for004.dat files
$!
$ IF FILE4.NES."" THEN  RENAME  'MOD''IDENT'.DUMP4;*  FOR004.DAT;*
$ CONTA:
$ GOTO CONTB
$ DEFPROG:
$!
$! DEFINE/NOLOG for004 and for009 files for KEEL or HULL
$!
$ DEFINE/NOLOG      FOR004          'PART1''IDENT'.DUMP4
$ DEFINE/NOLOG      FOR009          'PART1''IDENT'.DUMP9
$ CONTB:
$!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
$ IF PART1.NES."PULL".AND.PART1.NES."STAT" THEN GOTO NOPULL
$ OPEN/READ INFILE DUMPFILES.DAT                    'to dumpfiles.dat then read back
$                                                           'with plotpkg 0 output is 1
$ DEFINE NOLOG FOR016 'PROG'.PLOT                   ! rename for016 plot files
$ DEFINE/NOLOG FOR089 'PROG'.PLOT                   ! TECKCOLOUR produces for089
$!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
$NOPULL:
$ CALL PAGE
$ IF INTERACTIVE THEN CALL GETCPU  "''F$PROCESS()'" "START"
```

```
$ IF (EXT.EQS."MAINEXE") THEN CALL CHECK "``PROG``IDENT`.EXE"
$ IF (EXT.NES."MAINEXE") THEN CALL CHECK "``PROG``IDENT`.FOR"
$                                       ! make sure .FOR file is present
$ IF (EXT.EQS."MAINEXE") THEN GOTO CONT3  !don't compile or link if MAINEXE
$ ON SEVERE_ERROR THEN GOTO ERROR_COMPILE ! check for compilation errors
$ IF INTERACTIVE THEN ON WARNING THEN GOTO WARN_COMPILE
$                                       !check for COMPILE WARNING
$ CALL WRITMESS " COMPILING AND LINKING ``PROG` FOR PROB. ``IDENT`"
$ `COMPILEHULL` `PROG``IDENT` !compile assigned in HULL.COM
$ ON ERROR THEN CALL ERROR_COM          !if compilation error , report
$ GOTO CONT2                            ! if no errror then continue
$! compilation error has occured
$!---------------------------------------
$ ERROR_COMPILE:
$ CALL DEASS_PROG
$ CALL ERROR " ERROR IN COMPILING ``PROG``IDENT`"
$!---------------------------------------
$!---------------------------------------
$ WARN_COMPILE:
$ INQUIRE WARNCONT " Will you continue from this warning (Y/N)"
$ IF WARNCONT.EQ."Y" THEN GOT CONT2
$ IF WARNCONT.NES."N" THEN GOTO WARN_COMPILE
$ CALL DEASS_PROG
$ CALL ERROR " COMPILATION WARNING FOR ``PROG``IDENT`"
$!---------------------------------------
$ CONT2:
$ ON SEVERE_ERROR THEN GOTO ERROR_LINK
$ IF INTERACTIVE THEN ON WARNING THEN GOTO WARN_LINK !check for LINK WARNIN
$!
$!linkhull is assigned in HULL.COM ,if plotting then link with plotting routines
$ IF F$SEARCH("``PLOT_OBJ`").NES."" THEN PLOT_OBJ="," + PLOT_OBJ
$ IF (PART1.EQS."PULL".OR.PART1.EQS."STAT") THEN -
 `LINKHULL` `PROG``IDENT``PLOT_OBJ`
$! if not plotting then simply link problem with pre-assigned link options
$ IF (PART1.NES."PULL".AND.PART1.NES."STAT") THEN  `LINKHULL` `PROG``IDENT`
$ ON ERROR THEN CALL ERROR_COM
$ GOTO CONT3
$!---------------------------------------
$! linking error has occured
$ ERROR_LINK:
$ CALL DEASS_PROG
$ CALL ERROR " ERROR IN LINKING ``PROG``IDENT`"
$!---------------------------------------
$!---------------------------------------
$ WARN_LINK:
$ INQUIRE WARNCONT " Will you continue from this warning (Y/N)"
$ IF WARNCONT.EQ."Y" THEN GOT CONT3
$ IF WARNCONT.NES."N" THEN GOTO WARN_LINK
$ CALL DEASS_PROG
$ CALL ERROR " LINK WARNING FOR ``PROG``IDENT`"
$!---------------------------------------
$! Just before program is run , a process is spawned, where the command is held
$!in MONITOR_RUN , assigned in HULL.COM. Typically this is "MONITOR/PROC/TOPCPU"
$ CONT3:
$ ON ERROR THEN GOTO ERROR_RUN
$ IF (INTERACTIVE) THEN -
$ IF MONITOR_RUN.NES."" THEN SPAWN/PROCESS=`procname`/NOWAIT `monitor_run`
```

```
$ CALL WRITMESS " RUNNING ``PROG` FOR PROB. ``IDENT`"
$!-------------------------------------------------------
$! if not plotting then do not read dump-file file
$ IF PART1.NES."PULL".AND.PART1.NES."STAT" THEN GOTO RUNIT
$TOPR:
$ READ/END=ENDPLOT INFILE RECIN
$ DEFINE/NOLOG FOR004 FOR004.DAT;`RECIN`
$RUNIT:
$ DEFINE/USER_MODE/NOLOG SYS$INPUT SYS$COMMAND
$ RUN  `PROG``IDENT`
$ IF PART1.NES."PULL".AND.PART1.NES."STAT" THEN GOTO ENDRUN
$ GOTO TOPR                                          ! read next extension in file
$ENDPLOT:
$ CALL PROCESS_PLOTS
$ENDRUN:
$!-------------------------------------------------------
$ IF (INTERACTIVE) THEN -
$ IF MONITOR_RUN.NES."" THEN STOP `procname`
$ IF (F$SEARCH("OUTPUT.DAT").NE."") THEN RENAME OUTPUT.DAT `PROG``IDENT`.OUT
$ ON ERROR THEN CALL ERROR_COM
$ GOTO CONT4
$ ERROR_RUN:
$ IF (INTERACTIVE) THEN -
$ IF MONITOR_RUN.NES."" THEN STOP `procname`
$ CALL DEASS_PROG
$ CALL ERROR " ERROR IN RUNNING ``PROG` "
$ CONT4:
$ CALL DEASS_PROG
$ RET
$ ENDSUBROUTINE
$!*************************************************************************
$ DEASS_PROG: SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ IF INTERACTIVE THEN CALL GETCPU  "``F$PROCESS()`" "END"
$ DEASSIGN FOR060
$ DEASSIGN FOR005
$ DEASSIGN FOR006
$ IF PART1.EQS."HULL".OR.PART1.EQS."KEEL"  THEN GOTO DEFPROG
$  IF F$GETDVI("INFILE","EXISTS") THEN CLOSE INFILE
$ DEASSIGN FOR016
$ DEASSIGN FOR089
$ IF PART1.EQS."PULL" THEN GOTO CONT0
$  RENAME FOR004.DAT;* `MOD``IDENT`.DUMP9;*
$ GOTO CONTA
$ CONT0:
$  RENAME  FOR004.DAT;* `MOD``IDENT`.DUMP4;*
$ GOTO CONTA
$ DEFPROG:
$ DEASSIGN FOR004
$ DEASSIGN FOR009
$ IF PART1.EQS."KEEL".AND.F$SEARCH("KEEL``IDENT`.DUMP4").NES."" THEN -
  COPY KEEL`IDENT`.DUMP4 HULL`IDENT`.DUMP4;
$ IF PART1.EQS."KEEL".AND.F$SEARCH("KEEL``IDENT`.DUMP9").NES."" THEN -
  COPY KEEL`IDENT`.DUMP9 HULL`IDENT`.DUMP9;
$ IF PROG.NES."REZONE" THEN GOTO CONTA
$ IF F$SEARCH("FOR050.DAT").EQS."" THEN GOTO NO_REZONE
```

```
$ RENAME  FOR050.DAT; HULL'IDENT'.DUMP4;
$ CALL WRITMESS " NOTE  >> COPYING FOR050.DAT TO HULL''IDENT'.DAT; "
$ GOTO CONTA
$NO REZONE:
$ CALL ERROR " NO FOR050 FILE WAS PRODUCED FOR THE REZONE"
$CONTA:
$ RET
$ ENDSUBROUTINE
$!*****************************************************
$!SUBROUTINE : PROCESS PLOTS
$! This subroutine is called when any plotting program (PULL, STATION)
$!  is complete.
$!
$!
$!*****************************************************
$ PROCESS_PLOTS:SUBROUTINE
$ ON CONTROL_Y THEN CALL ERROR COM
$ ON ERROR THEN CALL ERROR " ERROR - COPYING PLOT FILES TO FOR016.DAT"
$ CHECKPLOT=F$SEARCH("*.PLOT;-1")
$ IF CHECKPLOT.EQS."" THEN GOTO ONE_PLOT
$ FILE_TIME=F$FILE_ATTRIBUTES("'CHECKPLOT'","CDT")
$ FILE_TIME:='F$CVTIME(FILE_TIME)'
$ IF FILE_TIME.LTS.BEGIN_TIME THEN GOTO ONE_PLOT
$ MANY_PLOT:
$ COPY/CONC/SINCE='SINCE_TIME' *.PLOT;* FOR016.DAT;
$ DELETE/SINCE='SINCE_TIME' *.PLOT;*
$! DIRECTORY/SINCE='START_TIME'/COLUM=1/OUTPUT=GMETA.DAT GMETA.CGM
$! METACAT
$ GOTO ENDIT
$ ONE_PLOT:
$ CHECKPLOT=F$SEARCH("*.PLOT;")
$ IF CHECKPLOT.EQS."" THEN CALL WRITMESS " No .PLOT files where produced"
$ IF CHECKPLOT.NES."" THEN RENAME *.PLOT; FOR016.DAT;
$ ENDIT:
$ ENDSUBROUTINE
```

```
$!*************************************************
$!                                  SUBROUTINE : GET_DUMPS
$! This subroutine is called when any plotting program (PULL, STATION)
$!  is complete.
$!
$!
$!*************************************************
$ GET_DUMPS:SUBROUTINE
$ ON ERROR THEN GOTO DUMPERR
$ ON CONTROL_Y THEN GOTO DUMPERR
$ CALL PAGE
$ CALL DRAW_LINES
$ CALL SCROLL_BOTTOM
$ IF (NODISPLAY) THEN GOTO NODISP
$ TYPE SYS$INPUT
        = = = = = = = = = = = = = = DUMP-FILE PLOTS = = = = = = = = = = = = = = =
    M-N       M is the version number of the first file,N  the last
    ALL       For all of the dump files
    START-N             The first dump file upto version n are used
    N-END   Version N up to the last dump
    LAST      For the last dump
    REPEAT To plot for the last dump-files plotted
    SKIPn      plot every nth file from the last one, n is an integer
    LASTn      plot the last n dump files . n is an integer

NOTE: Use these commands in combination, seperated by coma's (i.e. 1-3,5-9)
$ NODISP:
$!-------------------------------------------------------------
$ CALL SCROLL_TOP
$ IF PART1.EQS."PULL" THEN EXTENSION = "DUMP4"
$ IF PART1.EQS."STAT" THEN EXTENSION = "DUMP9"
$ DUMPNAME = "''MOD''IDENT'.''EXTENSION'"
$ FILEFILE = "DUMPFILES.DAT"                    ! file to hold list of dump files
$ BIGSTR: = 'P1'                                                ! BIGstr holds the option
$ IF (BIGSTR.EQS."".AND.(.NOT.INTERACTIVE)) THEN BIGSTR = "END"
$                                                               !default for this is "last"
$!-------------------------------------------------------------
$ CALL SCROLL_TOP
$START:
$ IF (BIGSTR.EQS."".AND.(INTERACTIVE)) THEN INQUIRE BIGSTR " Option ? "
$!-----------------------------------
$ BIGLOOP = -1
$ IF BIGSTR.EQS."REPEAT" THEN GOTO ENDALL
$ OPEN/WRITE INFILE DUMPFILES.DAT          ! open this file
$CONTA:
$ BIGLOOP = BIGLOOP + 1                    ! Break up bigstr into little strings
$ INSTR = F$ELEMENT(BIGLOOP,",",BIGSTR)  ! i.e. 1-2,5-6 goes to 1-2 then 5-6
$ IF INSTR.EQS."," THEN GOTO FIN
$!-----------------------------------
$ STRING0 = F$ELEMENT(0,"-",INSTR)         ! STRING0 holds first STRING of option
$ STRING1 = F$ELEMENT(1,"-",INSTR)         ! STRING1       second          "
$ TYPE0 = F$TYPE(STRING0)                  ! type0 = integer if STRING0 integer
$ TYPE1 = F$TYPE(STRING1)                  ! type1     "       "  STRING1   "
$ GOTO CONT0
$ERRIN:
$ IF (.NOT.INTERACTIVE) THEN GOTO ERR2  ! if batch then STOP
$ CALL WRITMESS " > > ERROR in directive ''BIGSTR' "
```

```
$ BIGSTR=""
$ GOTO START                                              ! prompt again
$ERR2:
$ CALL ERROR " >> ERROR IN DIRECTIVE ``INSTR`"
$ CALL ERROR_COM
$ EXIT
$DUMPERR:
$ CALL ERROR " ERROR IN DUMP_FILES SUBROUTINE"
$ CALL ERROR_COM
$ EXIT
$CONT0:
$ SKIPIT:="``F$EXTRACT(0,4,STRING0)`"        !
$ INCREMENT=1
$ JUMP=""
$ IF SKIPIT.EQS."SKIP" THEN INCREMENT=STRING0-"SKIP"
$ IF SKIPIT.EQS."LAST" THEN JUMP=STRING0-"LAST"
$ IF STRING0.EQS."LAST" THEN JUMP=""
$ FIRSTNUM="XXX"
$ LASTFILE=F$SEARCH("``DUMPNAME`;")     ! lastfile is the last dump file made
$ VER=F$ELEMENT(1,";",LASTFILE)         ! ver is the version number of this file
$ LASTNUM="YYY"                         ! TEST if integer later on
$ IF SKIPIT.EQS."SKIP" THEN LASTNUM=VER
$ IF SKIPIT.EQS."SKIP" THEN FIRSTNUM=1
$ IF STRING0.EQS."ALL" THEN LASTNUM=VER
$ IF STRING0.EQS."ALL" THEN FIRSTNUM=1
$ IF SKIPIT.EQS."LAST" THEN LASTNUM=VER
$ IF SKIPIT.EQS."LAST".AND.F$TYPE(JUMP).EQS."INTEGER" -
  THEN FIRSTNUM=%D`LASTNUM`-%D`JUMP`
$ IF STRING0.EQS."LAST" THEN LASTNUM=VER
$ IF STRING0.EQS."LAST" THEN FIRSTNUM=VER
$ IF STRING0.EQS."START" THEN FIRSTNUM=1
$ IF STRING1.EQS."END" THEN LASTNUM=VER
$ IF TYPE0.EQS."INTEGER" THEN FIRSTNUM=STRING0
$ IF TYPE0.EQS."INTEGER".AND.TYPE1.EQS."INTEGER" THEN GOTO ALLNUM
$ IF TYPE0.EQS."INTEGER".AND.STRING1.EQS."-" THEN LASTNUM=FIRSTNUM
$ IF TYPE1.EQS."INTEGER".AND.STRING1.LE.VER THEN LASTNUM=STRING1
$ IF TYPE1.EQS."INTEGER".AND.STRING1.GT.VER THEN LASTNUM=VER
$ GOTO CONT1
$ALLNUM:
$ IF STRING0.GT.STRING1 THEN  LASTNUM=STRING0
$ IF STRING0.GT.STRING1 THEN  FIRSTNUM=STRING1
$ IF STRING1.GT.STRING0 THEN  FIRSTNUM=STRING0
$ IF STRING1.GT.STRING0 THEN  LASTNUM=STRING1
$CONT1:
$ IF F$TYPE(FIRSTNUM).NES."INTEGER" THEN  GOTO ERRIN
$ IF F$TYPE(LASTNUM).NES."INTEGER" THEN  GOTO ERRIN
$ IF F$TYPE(INCREMENT).NES."INTEGER" THEN  GOTO ERRIN
$ IF FIRSTNUM.GT.LASTNUM THEN GOTO ERRIN
$ FLOOP=LASTNUM
$LOOP:
$ FILE=F$SEARCH("``DUMPNAME`;``FLOOP`")
$ IF FILE.EQS."".OR.FLOOP.EQ.0 THEN GOTO NOFILE
$ NUMOUT:=0000
$ NUMOUT[4-F$LENGTH(FLOOP),4]:=`FLOOP`
$ WRITE INFILE "``NUMOUT`"
$NOFILE:
$ FLOOP=%D`FLOOP`-%D`INCREMENT`
```

```
$ IF FLOOP.LT.FIRSTNUM THEN GOTO ENDIT
$ GOTO LOOP
$ENDIT:
$ GOTO CONTA
$FIN:
$  IF F$GETDVI("INFILE","EXISTS") THEN CLOSE INFILE
$ SORT/KEY=(POSITION:1,SIZE=4,DECIMAL,ASCEND)/NODUP DUMPFILES.DAT DUMPFIL
$ PURGE/K=1 DUMPFILES.DAT
$ENDALL:
$ RET
$ ENDSUBROUTINE
```

```
$!*****************************************************************************
$!                                  SUBROUTINE : GET_PROG
$! The name of the program to run may be held in PROG_TO_RUN . If not,
$! the name is prompted for in interactive mode , or the command file
$! terminated otherwise.
$!
$! REQUIRED PARAMETERS: PROG_TO_RUN
$!
$! PARAMETERS PRODUCED:       PROG  -  Either HULL , KEEL or PULL
$!                            EXT   -  Either SAIL or MAIN
$!*****************************************************************************
$ GET_PROG : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CALL PAGE
$ CALL DRAW_LINES
$ CALL SCROLL_BOTTOM
$!----------------------
$ NUMOPT = =5
$ OPT0: = = "PROG/KEEL/PULLKEEL/PULLHULL/HULL/STATHULL/REZONE"
$                                                    ! add to particular list ,
$ OPT1: = = "EXT/SAIL/MAIN/MAINEXE/PLANK"            ! or add your own option by following
$ OPT2: = = "NOEXE/NOEXE"                    ! sequence of options , and add variable
$ OPT3: = = "RUNBATCH/BATCH/DEFBATCH"
$ OPT4: = = "FLAGS/DEBUG/RECORD"
$!----------------------
$ ILOOP1 = -1                                ! Initialise variables that may be used
$ TOPINIT:
$ ILOOP1 = ILOOP1 + 1
$ IF ILOOP1.GT.NUMOPT-1  THEN GOTO ENDINIT
$ NEWVAR = F$ELEMENT(0,"/",OPT`ILOOP1`)
$ `NEWVAR`: = = ""
$ GOTO TOPINIT
$ENDINIT:
$!------------------------
$ IF (NODISPLAY) THEN GOTO NODISP
$ TYPE SYS$INPUT
        = = = = = = = = = = = = = = COMMAND FILE TO RUN HULL = = = = = = = = = = = =
 * Enter KEEL,HULL ,STATHULL,PULLHULL, PULLKEEL, REZONE

 * /PLANK    to just run plank
 * /SAIL     will run procedure from the Sail phase
       /MAIN      will compile, link then run the main program
       /MAINEXE  run the program without recompiling or linking
              ( i.e. simply use HULL/MAINEXE for restart run
 * /NOEXE if .FOR file is to be created but not run
 * /BATCH if  program is to be run in batch mode
 * /DEFBATCH  to run in batch mode, with default batch parameters
 * /RECORD to add additional info. to recordfile held in work dir.
 * /DEBUG to run (and compile/link if necessary) in DEBUG mode
$ NODISP:
$ CALL SCROLL_TOP
$ INP = "``PROG_TO_RUN`"
$!-------------------------------------------------
$! This part initialises the FLAG variables in varlist
$
$ TOPLOOP = 1
```

```
$ ENDOPT=NUMOPT-1
$ TOP0:
$ VARTOP=F$ELEMENT(`TOPLOOP`,"/",OPT`ENDOPT`)
$ IF (VARTOP.EQS."/") THEN GOTO TOP
$ `VARTOP`==0
$ TOPLOOP=TOPLOOP+1
$ GOTO TOP0
$!-----------------------------------------------------
$ TOP:
$ IF (INTERACTIVE.AND.(INP.EQS."")) -
 THEN INQUIRE INP " What is the program (KEEL,HULL,PULLKEEL etc.) "
$ TOP2:
$!-------------------
$ OPTVAR:=`INP`
$ OPTSTR:=`OPT0`
$ CALL INSTR "``OPTSTR`" "``OPTVAR`"
$ IF (RETVAL.EQS."") THEN GOTO ERROR1
$ PROG:==`RETVAL`
$ BATCH_STR==PROG                          ! Hold valid input string in batch_str
$ OPTVAR = OPTVAR - "/" - RETVAL
$ GOTO CONT0
$!-------------------
$! ----------------error has occured-----
$ ERROR1:
$ IF INTERACTIVE THEN GOTO CONT_ERR    ! else error
$ CALL ERROR " INPUT TO MAIN .COM FILE IS INCORRECT"        ! call error routine
$ CONT_ERR:                                                                    ! then promp
$ INQUIRE INP " What is the program (KEEL , HULL, PULLKEEL etc.)"
$ INP:=`INP`
$                                                                              ! and check a
$ GOTO TOP2
$!-------------------------------------------------------------
$ CONT0:
$ VARCOUNT=0
$ CONT:
$ VARCOUNT=VARCOUNT+1
$ COMM:=""
$ IF (VARCOUNT.NE.NUMOPT) THEN COMM:="OPTSTR=OPT``F$STRING(VARCOUNT)`"
$ `COMM`
$ VARNAME==F$ELEMENT(0,"/",OPTSTR)
$ `VARNAME`==""
$ IF ((OPTVAR.NES."").AND.(VARCOUNT.EQ.NUMOPT))  THEN GOTO  ERROR2
$ IF (OPTVAR.EQS."")  THEN GOTO CONT3
$ CALL INSTR "``OPTSTR`" "``OPTVAR`"
$ IF (RETVAL.EQS."") THEN GOTO CONT
$ IF (VARCOUNT.LT.(NUMOPT-1)) THEN GOTO LTNUMOPT
$ COMM=" ``RETVAL`==1"
$ `COMM`
$ GOTO CONTZ
$LTNUMOPT:
$ COMM=" ``VARNAME`==RETVAL"
$ `COMM`
$ IF RETVAL.NES."BATCH".AND.RETVAL.NES."DEFBATCH".AND.RETVAL.NES."RECORD"
  THEN BATCH_STR==BATCH_STR+"/"+RETVAL
$ !Add valid input string to batch_str
$CONTZ:
$ OPTVAR = OPTVAR- "/" - RETVAL
```

```
$! OPTVAR:==`OPTVAR`
$ IF (OPTVAR.EQS."") THEN GOTO CONT3
$ GOTO CONT
$!-----------------------------------------------------------------
$! ----------------error has occured-----
$ ERROR2:
$ !WRITE SYS$OUTPUT "OPTVAR=``OPTVAR` VARNAME=``VARNAME`"
$ IF INTERACTIVE THEN GOTO ERROR3
$ CALL ERROR " DIRECTIVE ``OPTVAR` TO .COM FILE IS INCORRECT"
$!--------------------------------------
$ ERROR3:
$ CALL WRITMESS " >> ERROR IN DIRECTIVE ``OPTVAR`<<"
$ INQUIRE OPTVAR " What was the desired directive (return for none) "
$ OPTVAR:=`OPTVAR`
$ IF (OPTVAR.NES."") THEN GOTO CONT0
$ CONT3:
$ RET
$ ENDSUBROUTINE
```

```
$!*****************************************************************************
$!                                  SUBROUTINE  INSTR
$!
$! PARAMETERS REQUIRED : INSTR,INVAR
$!*****************************************************************************
$ INSTR:SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ INSTR:=`P1`
$ INVAR:=`P2`
$ ILOOP0=0
$ TESTSTR0:=`F$ELEMENT(ILOOP0,"/",INVAR)`
$ LOOP0:
$ ILOOP=1
$ TESTSTR:=`F$ELEMENT(ILOOP,"/",INSTR)`
$ LOOP:
$ IF (TESTSTR.EQS.TESTSTR0) THEN GOTO ENDINSTR
$ ILOOP=ILOOP+1
$ TESTSTR:=`F$ELEMENT(ILOOP,"/",INSTR)`
$ IF TESTSTR.NES."/" THEN GOTO LOOP
$ ILOOP0=ILOOP0+1
$ TESTSTR0:=`F$ELEMENT(ILOOP0,"/",INVAR)`
$ IF (TESTSTR0.NES."/") THEN GOTO LOOP0
$ RETVAL:==""
$ GOTO RETSTR
$ ENDINSTR:
$ RETVAL:==`TESTSTR`
$ RETSTR:
$ RET
$ ENDSUBROUTINE
```

```
$!*****************************************************************************
$!                                SUBROUTINE : GET_MAIN_INPUT
$! Gets the name of the main input file for this specific run
$! PARAMETERS REQUIRED : INTERACTIVE , MAIN_INPUT
$!*****************************************************************************
$ GET_MAIN_INPUT: SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CALL PAGE
$ CALL DRAW_LINES
$ CALL SCROLL_BOTTOM
$!-----------------------------------------------
$ IF (NODISPLAY) THEN GOTO NODISP
$ TYPE SYS$INPUT
        = = = = = = = = = = = MAIN INPUT FILE TO RUN HULL = = = = = = = = = = = = =


  *     The user input file to be used for HULL must reside in the
        WORK directory set up in the HULL.COM file

            If a new run is required with a changed input file , this
        new file must be in the WORK directory NOT in the sub-directory
        for the problem

$ NODISP:
$ CALL SCROLL_TOP
$!-----------------------------------------------
$ TOP:
$ IF (INTERACTIVE.AND.(MAIN_INPUT.EQS."")) THEN -
  INQUIRE MAIN_INPUT "What is the main input file"
$ TOP2:
$  FILE=F$SEARCH(MAIN_INPUT)                          ! Is the file present
$ IF (FILE.NES."") THEN GOTO CONT                     ! if so continue
$!------------ error has occured , file not found--
$ IF .NOT.INTERACTIVE THEN GOTO CONT_ERR              ! if not interactive end
$  INQUIRE MAIN_INPUT "What is the main input file"   ! else inquire again
$ GOTO TOP2                                                   ! then check again
$ CONT_ERR:
$                                                             ! file not found ,
$ CALL ERROR "Main input file not found"          ! finish
$!-----------------------------------------------
$ CONT:
$ FILE_WITH_NO_EXTENSION=F$ELEMENT(0,".",MAIN_INPUT)
$ NEW_INPUT"=="'FILE_WITH_NO_EXTENSION'                ! make MAIN_INPUT global
$ MAIN_INPUT"=="'MAIN_INPUT'                           ! make MAIN_INPUT global
$ RET
$ ENDSUBROUTINE
```

```
$!*****************************************************************************
$!                                  SUBROUTINE : ERROR_COM
$!
$!            This sub. is an emergency exit to a bad command file
$!             it will DEASSIGN  all definitions and close relevant files
$!             , then it will attempt to "reboot" the system by the command
$!              @[`ROOT_DIR`]hull.com.The current directory is then set
$!       as the default directory.
$!-----------------------------------------------------------------------
$ ERROR_COM : SUBROUTINE
$ ON ERROR THEN CALL PUT_LINE "6" "22" " Problem executing error subroutine"
$             CALL WRITMESS " ERROR WHILST EXECUTING COMMAND PROCEDURE"
$              IF F$GETDVI("MESSF","EXISTS") THEN CLOSE MESSF
$!            DEASS/ALL                                    ! deassign all definitions
$             @`ROOT_DISK`[`ROOT_DIR`]HULL.COM
$ WRITE SYS$OUTPUT -
 " HULL.COM REBOOTED . NOW IN ``WORK_DISK`[``WORK_DIR`.RUN``IDENT`]"
$ CALL FINISH
$ ENDSUBROUTINE
$!*****************************************************************************
$!                                  SUBROUTINE WRITMESS :
$!  The  message to be written to the dayfile file should have beem passed
$!  as a parameter . P1 is written to dayfile , then written to sys$output
$!
$! REQUIRED PARAMETERS : -message-
$!*****************************************************************************
$!                    subroutine error
$ WRITMESS : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ IF P2.EQS."" THEN P2=30
$ IF VT100 THEN          CALL PUT_LINE "6" "``P2`" "``P1`"
$ IF (VT100.NE.1) THEN  WRITE SYS$OUTPUT "``P1`"
$       WRITE MESSF "``P1`"
$!       WRITE sys$output "``P1`"
$ RET
$ ENDSUBROUTINE
$!
$!*****************************************************************************
$!                                  SUBROUTINE ERROR :
$!  The error message to be written to the dayfile file should have beem written
$! to variable MESS . MESS is written to dayfile , then the FINISH routine
$! is called to close relevant files etc.
$!
$! REQUIRED PARAMETERS : MESS
$!*****************************************************************************
$!                    subroutine error
$ ERROR: SUBROUTINE
$       WRITE MESSF "``P1`"
$             WRITE MESSF "Command routine status.  "F$MESSAGE($STATUS)"
$       WRITE sys$output "``P1`"
$ CALL PUT_LINE "22" "1" "``P1`"
$SET NOON
$ IF (INTERACTIVE) THEN STOP MON_RUN
$SET ON
$             CALL FINISH
$ RET
```

```
$ ENDSUBROUTINE
$!
$!*******************************************************************************
$!                                   SUBROUTINE FINISH :
$! This subroutine closes the dayfile , then copies it from then directory
$! holding the dayfile to "dayfile."ident.
$!*******************************************************************************
$ FINISH: SUBROUTINE
$ ON ERROR THEN WRITE SYS$OUTPUT " ERROR IN FINISH ROUTINE"
$ ON CONTROL_Y THEN WRITE SYS$OUTPUT " CONTROL_Y DISABLED"
$ SET NOON
$ CALL SCROLL 1 24
$ ASSIGN JUNK.DAT SYS$OUTPUT
$ IF F$GETDVI("INFILE","EXISTS") THEN CLOSE INFILE
$ IF F$GETDVI("MESSF","EXISTS") THEN CLOSE MESSF
$ IF F$GETDVI("BFILE","EXISTS") THEN CLOSE BFILE
$ IF F$GETDVI("SFILE","EXISTS") THEN CLOSE SFILE
$ IF F$GETDVI("IN","EXISTS")    THEN CLOSE IN
$ IF F$GETDVI("OUT","EXISTS")   THEN CLOSE OUT
$ IF F$GETDVI("RECF","EXISTS")  THEN CLOSE RECF
$ IF F$GETDVI("INTERM","EXISTS") THEN CLOSE INTERM
$ NEWFILE="''F$DIRECTORY()'DAYFILE.''IDENT'"
$ RENAME DAYFILE `NEWFILE`                    ! copy dayfile from work directory
$                                                     ! to current dir
$!---------------------------------------------
$ FILE4=F$SEARCH("FOR004.DAT")
$ FILE9=F$SEARCH("FOR009.DAT")
$ IF FILE4.NES."".AND.PART1.EQS."PULL"  THEN -
  RENAME FOR004.DAT;* 'MOD''IDENT'.DUMP4;*
$ IF FILE4.NES."".AND.PART1.EQS."STAT"  THEN -
  RENAME FOR004.DAT;* 'MOD''IDENT'.DUMP9;*
$ IF FILE9.NES.""  THEN RENAME FOR009.DAT;* 'MOD''IDENT'.DUMP9;*
$!---------------------------------------------
$ CALL PURGE_FILE "*.DIAG"
$! CALL PURGE_FILE "DAYFILE.*"
$! CALL PURGE_FILE "*.EXE"
$! CALL PURGE_FILE "*.''IDENT'"
$! CALL PURGE_FILE "*.COM"
$ DEASSIGN SYS$OUTPUT
$ DELETE JUNK.DAT;
$ DEASSIGN SYS$ERROR
$ IF F$SEARCH("ERRORS.DAT").EQS."" THEN GOTO NOERRORS
$ IF F$FILE_ATTRIBUTES("ERRORS.DAT","EOF").EQS."0" THEN GOTO DEL_ERRORS
$ IF INTERACTIVE THEN CONVERT/APPEND ERRORS.DAT `NEWFILE`
$ IF .NOT.INTERACTIVE.AND.F$SEARCH("LOGILE.''IDENT'").NES."" THEN -
    CONVERT/APPEND LOGFILE.''IDENT' `NEWFILE`
$DEL_ERRORS:
$ DELETE ERRORS.DAT;
$NOERRORS:
$ SET ON
$  STOP
$ ENDSUBROUTINE
$!
$!
$!*******************************************************************************
$!                                   SUBROUTINE PURGE_FILE
$! The name of the file to be PURGED is checked for existence.If the file
```

```
$! is found then a purge with a /k=1 is done.If it is not found then
$! no further action is taken
$!
$! REQUIRED PARAMETERS : P1
$!*******************************************************************************
$ PURGE_FILE: SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CHECK_FILE:='P1'
$ FILE=F$SEARCH(CHECK_FILE)                    ! look for file in current dir
$ IF (FILE.EQS."") THEN GOTO CONT              !   if found exit subroutine
$ PURGE/KEEP=2 'CHECK_FILE'
$ CONT:
$ RET
$ ENDSUBROUTINE
$!
$!
$!*******************************************************************************
$!                                SUBROUTINE CHECK :
$! The name of the file to be checked is held in check_file .If the file
$! is found then no action is taken.If it is not found then an error message
$! is written and the error subroutine is called
$!
$! REQUIRED PARAMETERS : P1
$!*******************************************************************************
$ CHECK : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ CHECK_FILE:='P1'
$ FILE=F$SEARCH(CHECK_FILE)                              ! look for file in current dir
$ IF (FILE.NES."") THEN GOTO CONT                        !   if found exit subroutine
$ CALL ERROR " ERROR - ''CHECK_FILE' NOT FOUND"          !   and call error routine
$ CONT:
$ RET
$ ENDSUBROUTINE
$!
$!*******************************************************************************
$!                                SUBROUTINE GETCPU :
$!
$! REQUIRED PARAMETERS : P1
$! this subroutine attempts to get system details about resources used for run
$!*******************************************************************************
$ GETCPU : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ PROC:='P1'                                             ! name of process
$ PID=F$PID(PROC)                                ! get number for this process
$ IF PID.EQS."" THEN GOTO CONT                           ! if non skip code
$ IF P2.EQS."END" THEN GOTO ENDCPU
$ CPUTIME==F$GETJPI("''PID'","CPUTIM")  ! get cpu time for process
$ IOTIME==F$GETJPI("''PID'","DIRIO")
$ GOTO CONT
$ ENDCPU:
$ CPUTIME=F$GETJPI("''PID'","CPUTIM") - CPUTIME    ! get cpu time for process
$ IOTIME=F$GETJPI("''PID'","DIRIO") - IOTIME
$ CPUTIME=CPUTIME/100
$ HOURS=CPUTIME/3600
```

```
$ MINS=(CPUTIME - (3600*HOURS))/60
$ SECS=CPUTIME - (3600*HOURS) - (60*MINS)
$ CALL WRITMESS " ****************************************"
$ CALL WRITMESS -
 " CPU TIME FOR ``PROG` IS ``HOURS`:HOURS,``MINS`:MINS,``SECS`:SECS"
$ CALL WRITMESS " DIRECT IO COUNT  IS ``IOTIME`"
$ CALL WRITMESS " ****************************************"
$ CONT:
$ RET
$ ENDSUBROUTINE
```

```
$!*********************************************************************
$ PUT_LINE : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ IF (VT100.NE.1) THEN GOTO NO_VT100
$ LINE = 'P1'                          ! line to display text
$ COL = 'P2'                           ! column to display text
$ TEXT = "''P3'"          ! text to be displayed
$ WRITE SYS$OUTPUT " "
$ WRITE SYS$OUTPUT ""
$!              puts the cursor at (lin,col) on the screen of the VT-100
$ BLANK=''
$ WRITE SYS$OUTPUT "INE';''COL'H''BLANK'"
$ WRITE SYS$OUTPUT "INE';''COL'H''TEXT'"
$ WRITE SYS$OUTPUT " "
$ IF SCRTOP.EQ.1 THEN CALL SCROLL_TOP
$ IF SCRTOP.EQ.0 THEN CALL SCROLL_BOTTOM
$ NO_VT100:
$ ENDSUBROUTINE
$!*********************************************************************
$ DRAW_LINES : SUBROUTINE
$ IF (VT100.NE.1) THEN GOTO NO_VT100
$ CALL PUT_LINE 6 1 "STATUS OF PROGRAM>>"          ! standard scren format
$ NO_VT100:
$ ENDSUBROUTINE
$!*********************************************************************
$ SCROLL : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ IF (VT100.NE.1) THEN GOTO NO_VT100       ! skip code if non vt100
$!
$!              sets scrolling region on VT100 from lstart to lend
$!
$ LSTART='P1'                          ! start line of region
$ LEND='P2'                            ! end line of scroll region
$!
$ WRITE SYS$OUTPUT "START';''LEND'r"      ! scroll command sent to term
$! This puts cursor at the top of the scroll region to be made
$ WRITE SYS$OUTPUT "START';1H
$ NO_VT100:
$ ENDSUBROUTINE
$!*********************************************************************
$ SCROLL_BOTTOM : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ IF (VT100.NE.1) THEN GOTO NO_VT100
$!
$!              sets scrolling region on VT100 from lstart to lend
$!
$ SCRTOP==0                                    ! set scroll region flag
$ WRITE SYS$OUTPUT " "
$ CALL SCROLL 8 24                     ! call scroll subroutine
$ NO_VT100:
$ ENDSUBROUTINE
$!*********************************************************************
$ SCROLL_TOP : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
```

```
$ ON CONTROL_Y THEN CALL ERROR_COM
$ IF (VT100.NE.1) THEN GOTO NO_VT100
$!
$!              sets scrolling region on VT100 from lstart to lend
$!
$ SCRTOP == 1                                   ! set flag to show scroll region
$ WRITE SYS$OUTPUT " "
$ CALL SCROLL 1 4                       ! call scroll subroutine
$                                               ! clear the first four lines
$ WRITE SYS$OUTPUT "
$ WRITE SYS$OUTPUT "
$ WRITE SYS$OUTPUT "
$ WRITE SYS$OUTPUT "
$ NO_VT100:
$ ENDSUBROUTINE
$!*****************************************************************
$ PAGE : SUBROUTINE
$ ON ERROR THEN CALL ERROR_COM
$ ON CONTROL_Y THEN CALL ERROR_COM
$ IF (VT100.NE.1) THEN GOTO NO_VT100        ! don't format for non vt100's
$!
$!              homes cursor and clears the screen on the VT-100
$!
$ WRITE SYS$OUTPUT " "
$ CALL SCROLL 1 24                              ! first reset scroll region
$ WRITE SYS$OUTPUT ""           ! this clears the screen
$ NO_VT100:
$ ENDSUBROUTINE
```

Appendix B: Listing of a typical HULL.COM

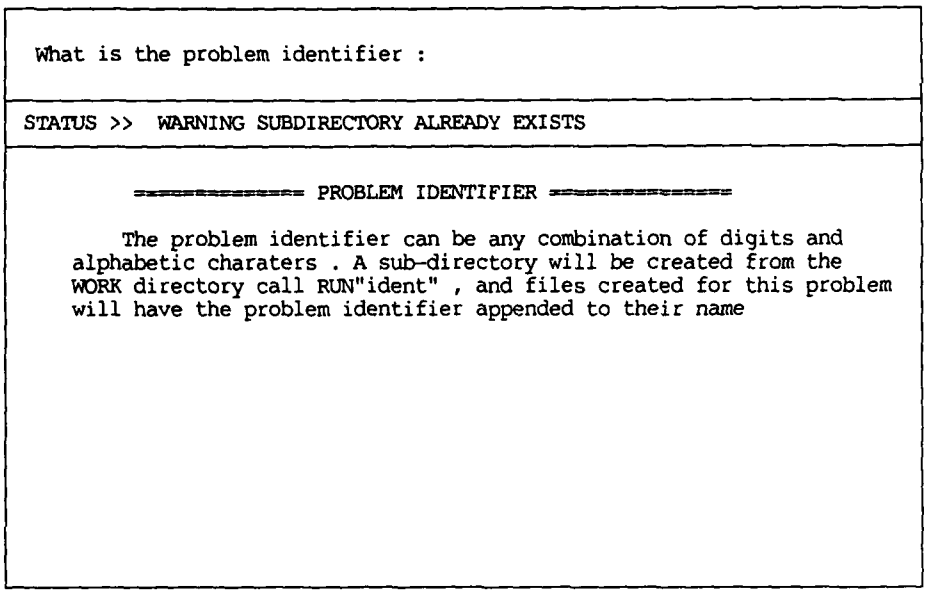
```
$ CALL SET_ROOT
$!==============================================================
$ WORK_DIR:==fred.hull.work
$ WORK_DISK:==diska:
$ DEFINE/NOLOG      HULL_LIB  diska:[harry.source]hull121.lib:
$ DEFINE/NOLOG      PLANK_EXE      diska:[sam.progs]plank.exe
$ DEFINE/NOLOG      SAIL_EXE  [harry.source]sail.exe
$ DEFINE/NOLOG      MAT_LIB   diskb:[fred.files]matlib.121
$ DEFINE/NOLOG      DAYFILE   ['WORK_DIR']DAYFILE.DAT:
$ PLOT_OBJ:==[fred.progs]calcomp.obj
$! SET VT100=1 FOR VT100 COMPATIBLE OR VT100=0 IF NOT and 3 FOR NO MENUS
$ VT100==1
$! cost_list:=="" if a cost code is not required at top of batch file
$ COST_LIST:==0001/0003/0009
$ BATCH_LIST:==ext$batch/slow$batch/sys$batch
$ LINKHULL :== link/nomap
$ COMPILEHULL :== for/nolis
$ MONITOR_RUN:== mon proc/topc
$ RUN_HULL:== @[fred.hull.commmands]HULL_JOB.COM:
$EXIT
$!==============================================================
$SET_ROOT:SUBROUTINE
$ WHERE=F$ENVIROMENT("PROCEDURE")
$! This little bit of code should set up the root_disk and dir to
$! the disk and dir where this file exists
$ ROOT_DISK==F$ELEMENT(0,"[",WHERE)
$ ROOT_DIR==F$ELEMENT(0,"]",WHERE) - ROOT_DISK - "["
$ENDSUBROUTINE
```

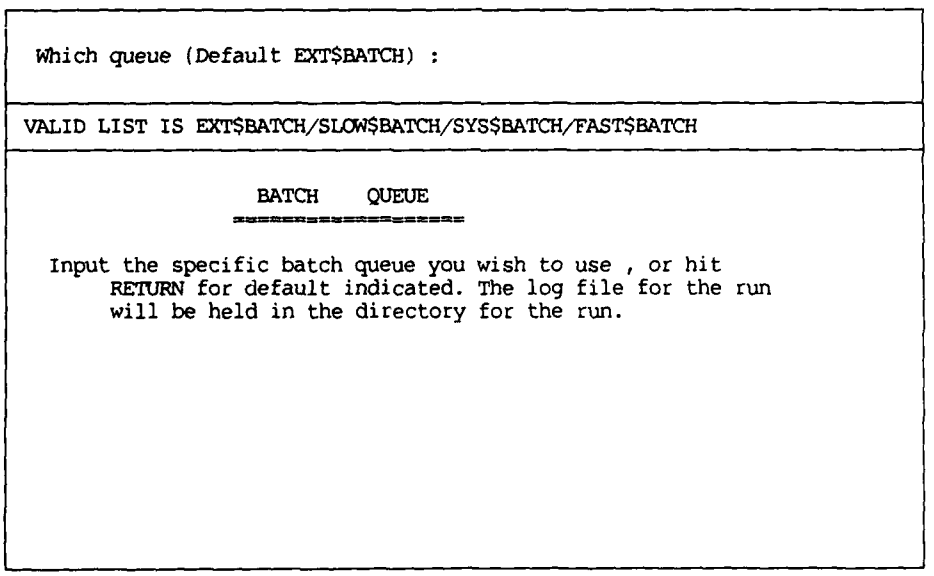
```
Option :

STATUS >>

        ============= COMMAND FILE TO RUN HULL =============

  * Enter KEEL,HULL ,STATHULL,PULLHULL, PULLKEEL, REZONE

  * /PLANK     to just run plank
  * /SAIL      will run procedure from the Sail phase
        /MAIN      will compile, link then run the main program
        /MAINEXE   run the program without recompiling or linking
                   ( i.e. simply use HULL/MAINEXE for restart run

  * Add /NOEXE if .FOR file is to be created but not run
  * Add /BATCH if  program is to be run in batch mode
  * Add /DEFBATCH  to run in batch mode, with default batch parameters
  * Add /DEBUG to compile,link and run program in debug mode
  * Add /RECORD to add lines to record file
```

```
What is the Main input file :

STATUS >>

        ========== MAIN INPUT FILE TO RUN HULL ==========


  *     The user input file to be used for HULL must reside in the
        WORK directory set up in the HULL.COM file

            If a new run is required with a changed input file , this
        new file must be in the WORK directory NOT in the sub-directory
        for the problem
```

```
What is the cost-code (Default 2180)

VALID LIST IS 2180/2230/2350
```

```
Option ? :

STATUS >>

          ============== DUMP-FILE PLOTS ===============
    M-N   M is the version number of the first file,N  the last
    ALL   For all of the dump files
    START-N       The first dump file upto version n are used
    N-END Version N up to the last dump
    LAST  For the last dump
    REPEAT        To plot for the last dump-files plotted
    SKIPn      plot every nth file from the last one, n is an integer
    LASTn      plot the last n dump files , n is an integer

NOTE: Use these commands in combination, seperated by coma's (i.e.1-3,5-9)
```


What is the problem identifier :
STATUS >> WARNING SUBDIRECTORY ALREADY EXISTS
PROBLEM IDENTIFIER
The problem identifier can be any combination of digits and alphabetic charaters . A sub-directory will be created from the WORK directory call RUN"ident" , and files created for this problem will have the problem identifier appended to their name


Which queue (Default EXT$BATCH) :
VALID LIST IS EXT$BATCH/SLOW$BATCH/SYS$BATCH/FAST$BATCH
BATCH QUEUE
Input the specific batch queue you wish to use , or hit RETURN for default indicated. The log file for the run will be held in the directory for the run.

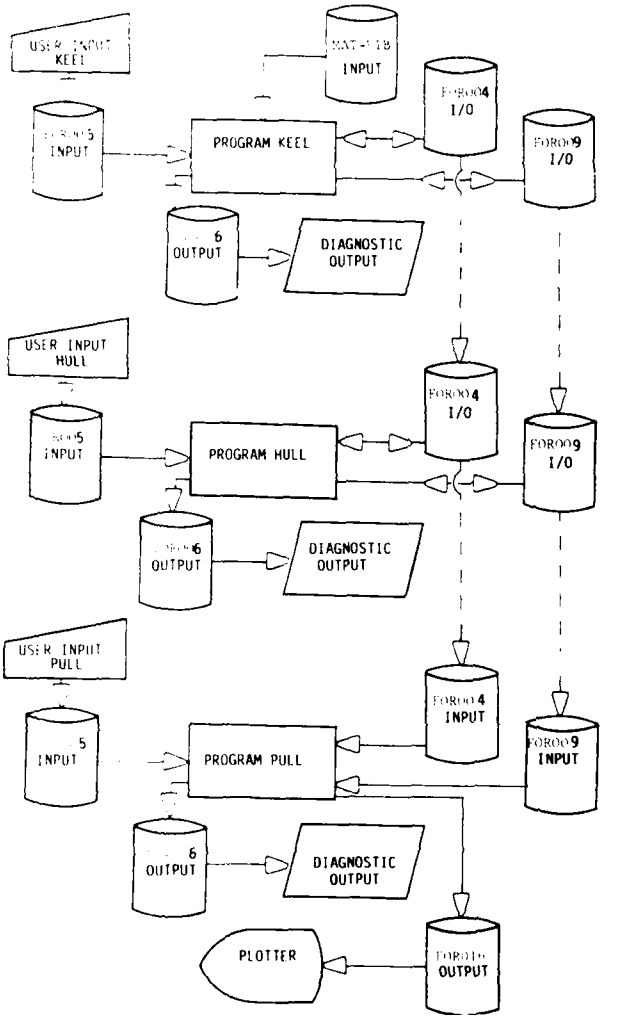


FIGURE 1 File organisation and exchange for main HULL programs

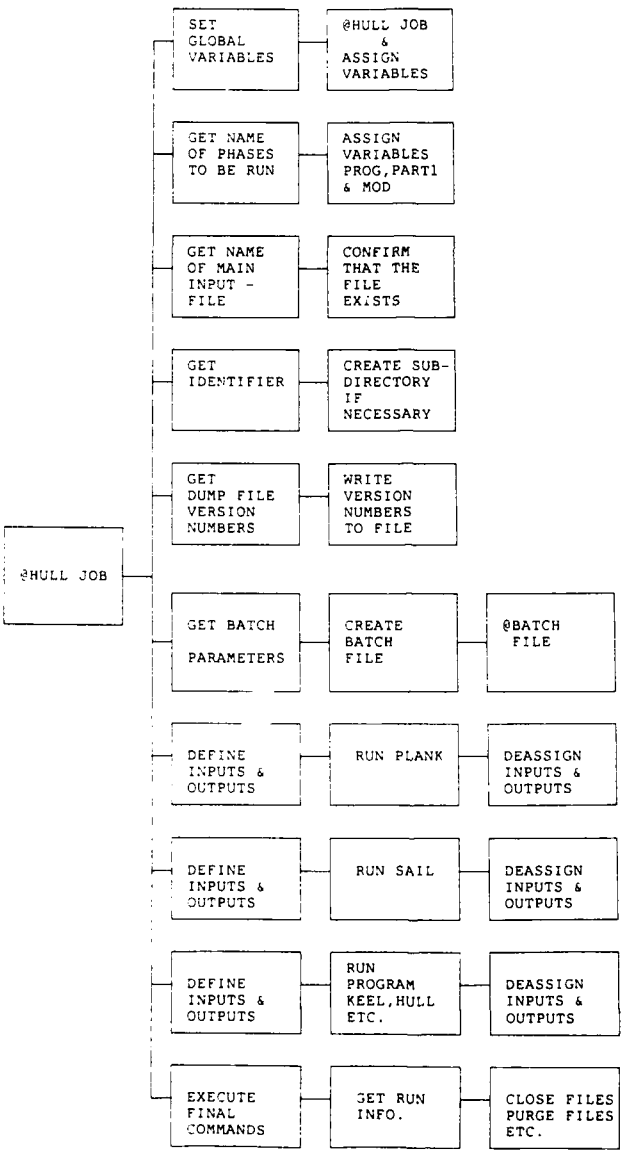


FIGURE 2 Functional Description

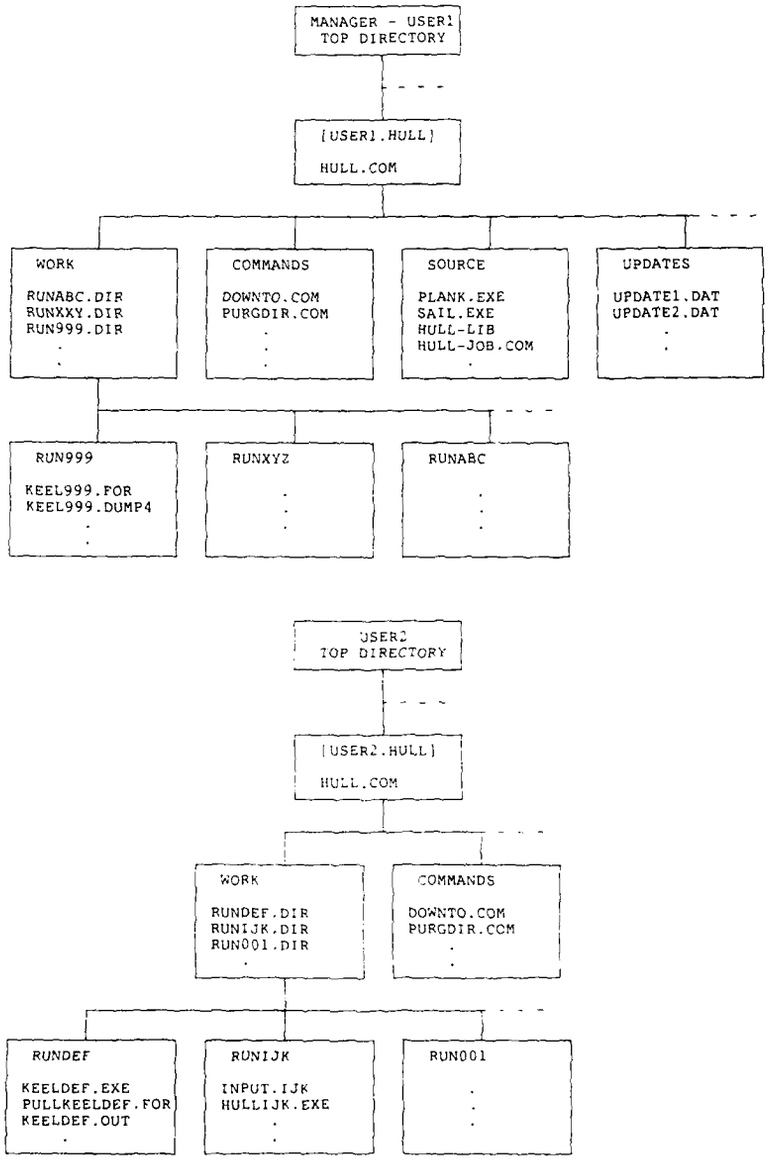


FIGURE 3 Directory set-up for a HULL system

SECURITY CLASSIFICATION OF THIS PAGE **UNCLASSIFIED**

DOCUMENT CONTROL DATA SHEET

REPORT NO.	AR NO.	REPORT SECURITY CLASSIFICATION
MRL-GD-0023	AR-005-710	Unclassified

TITLE

The procedures for easy running of the HULL computer code under VMS

AUTHOR(S)

Ross J. Kummer

CORPORATE AUTHOR

DSTO, Materials Research Laboratory
PO Box 50,
ASCOT VALE VIC 3032

REPORT DATE	TASK NO.	SPONSOR
July 1989	DST 88/112	DSTO

FILE NO.	REFERENCES	PAGES
G6 4 8 3628	149	54

CLASSIFICATION/LIMITATION REVIEW DATE

CLASSIFICATION/RELEASE AUTHORITY
Chief, Explosives Division
MRL

SECONDARY DISTRIBUTION

Approved for public release

ANNOUNCEMENT

Announcement of this report is unlimited

KEYWORDS

Hull code
DCL
VAX
Hydrodynamic flow

SUBJECT GROUPS 0062B

ABSTRACT

Version 121 of the HULL computer code has been installed on a VAX 8700 at MRL. This report explains the complex command procedures that were written to provide an easy interface for running HULL on the VAX system. Full instruction guides are included, with installation details to provide easy implementation of the procedures at other VAX sites.